"INFORCOM"

MMIIRR LANDS

Circle Lebedev

1995

PDF version by Deny (Deniseno D.A.)
e-mail: DenyDA@mail.ru
2009

ENTERPRISE FROM PDF'ika.

A lot of people who read these lines think where this book came from, didn't they? And You'll be right. Partly.

At the beginning, I downloaded txt file with Virtual TR-DOS, on the news for 11.03.2006. Figures below:

@ Another update under Books. Okay, Oleg Dagtyre sent a book of fascinating games. On a domestic computer, and Slave Kalinin dug a book of the World of Sounds. Spectrum. The author of the last book on his website claims that the manuscript was in time. Ambassador Inforcoma, but he never saw the light.

The author's wrong. The book was printed, but electronically in ZX-FORUM No. 4, really, no. A picture. I've decided to make this book in print, that's PDF. Total 2 years of work (more abandoned) You'll see for yourself.

Deny (DenyDA@mail.ru)

CONTENTS

PREDICTION

This book is intended for the readers who decided to write programs and want to design them at high level, which is impossible without using sound capabilities of ZX Spectrum. It is written from the simple to the complex, and even a not very well trained person will be able to understand it. The book can also serve as a primer on assembly language programming, because it is literally full of examples.

It contains practically all the information you may ever need to create soundtracks for both game and system programs.

The book covers only the programming of standard Spectrum sound devices (the #FE port and the musical co-processor). Such exotics as DAC, ADC, M IDI, inherent to Spectrum monsters (AT M TURBO, PROFI) and some self-made devices are not even touched because of its non-standard nature. Although, with the DAC (Digital to Analog Converter) alone it would be possible to create almost fantastic sounds. Perhaps sometime in the future, I will describe these means of sound creation.

Now about what it does contain.

The first chapter describes the nature of sound and its characteristics.

The next three chapters describe the Spectrum BASIC operator VEER and how to create effects and melodies with it.

The next chapter describes how sound is produced in ZX-Spectrum and port 254(#FE).

The following is a story about creating effects and music in machine code.

Then about the processing of signals fed to the tape input of your computer, and about programs that synthesize human speech.

The next chapter talks about sound on interrupts.

The second part of the book is about the AY-3-8910 (AY-3-8912) music coprocessor.

First, the BASIC-128 PLAY operator and everything related to it is described.

The following chapter will help you understand the coprocessor control and its registers. With its help you will learn how to program the AY chip in codes.

Finally, there is a software overview, which includes various music editors, sound effects editors, speech synthesizers, demonstration programs, etc. Three such programs are described in detail. These are SUPER SOUND, a sound effects editor; W HA M THE M USIC BOX, a music editor; and ASC SOUND M ASTER (AS M), a music editor for the coprocessor.

Appendix 1 contains the listings of all the sound effects of SUPER SOUND, and Appendix 2 gives tips for using the GENS4 assembler.

The notation used in the book: CS - key Caps Shift;

SS - Symbol Shift key;

CS/1 - simultaneous pressing of keys (in this case Caps Shift and 1); A+2 - sequential pressing of keys (in this case A and 2);

[1] - the publication number in the reference list, at the end of the book.

1. Physics of Sound.

Scientifically speaking, sound is the mechanical longitudinal vibrations of an elastic medium. In the language of mere mortals, it's just the vibrations of the air.

The human ear is very sensitive to changes in air pressure; it picks up the slightest fluctuation in air pressure.

Humans can perceive sound vibrations with frequencies ranging from 20 Hz to 20,000 Hz (Hz is hertz. One vibration per second). A sound with a frequency of less than 20 Hz is called infrasound, and over 20,000 Hz is called ultrasound.

The higher the frequency, the higher the audible sound. The ear's sensitivity is not the same across the entire range of sound frequencies. The lowest and highest frequencies are much less perceptible than the middle frequencies.

The loudness of a sound directly depends on the amplitude of the vibrations and is measured in decibels (dB). The greater the amplitude, the louder the audible sound. A person can perceive a sound as loud as 0 to 130 dB. The loudest sounds (100 - 130 dB) are perceived painfully, and at a volume greater than 130 dB the eardrums in the ears burst and the person becomes deaf.

The next characteristic of sound is timbre. Timbre is the shape of a sound wave. Because our ears are able to distinguish timbres, we can distinguish the sound of a guitar from a piano. To illustrate this, take a look at Figure 1. It shows two sound waves with different timbres.

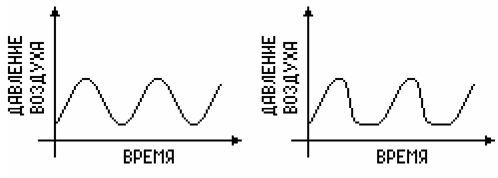


Рис. 1 Звуковые тембры.

Another characteristic is periodicity. If a sound wave consists of a single repeating fragment, then the sound can be considered periodic, otherwise it is not. A periodic sound is perceived as a musical tone, while a non-periodic sound is perceived as noise. There are several different kinds of noise. The two most common are white noise and impulse noise. Their diagrams are shown in Figure 2.

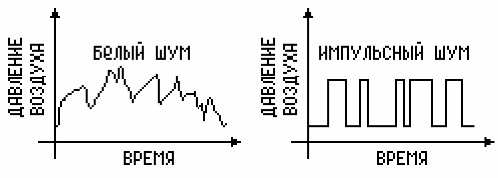


Рис. 2 Белый и импульсный шумы.

The last characteristic of sound is duration. It is created artificially, because it determines not the sound itself, but the time of its sounding. This characteristic is mostly used in music.

All of the above sound parameters are quite relative, because each person feels them differently. For example, the same sound may seem too loud to one person and too quiet to another.

2. Operator VEER.

The only way to get sound that Spectrum Basic provides is with the VER operator. Its format is:

```
WEER t,f
```

- where t is the duration of the sound in seconds, and f is the pitch in semitones. To raise or lower a note by an octave, its value must be increased or decreased by 12, respectively. So

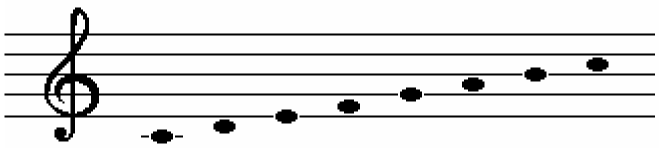
```
BEEP 0.5.0
```

will play the note BEFORE the first octave for half a second, and

```
BEEP 0.5,12
```

- the same note in B, but in the second octave.

This notation is not very accurate, so take a look at Fig. 3. There, the top line shows the approximate and the bottom line shows the exact parameters of the VEER operator for the first octave.



	До	Ре	Ми	Фа	Соль	Ля	Си	До
Прибл.		2	4	5	7	9	11	12
Точно	0	2.039	3.86	4.98	7.02	8.84	10.88	12

Рис. 3 Параметры BEEP для первой октавы.

The musical range of BEEP is quite wide, from -60 to 69. It covers 18.5 octaves, but the lowest and highest octaves are of no practical interest for music creation, although they may well be used in sound effects.

And in conclusion, we can add about the convenience that Spectrum-Beissick gives us in writing decimal fractions. If such a fraction is less than one, we don't need to put zero before the point. Thus

```
BEEP 0.125.11
```

can be written as

```
BEEP .125.11.
```

That's exactly what I'm going to do from here on in the book.

2.1. Creating effects on VEER.

In order to make something more interesting than a simple beep, the VER operator is usually included in some loop where one or both of its parameters are changed. The simplest example might look like this:

```
10 FOR A=0 TO 60
20 BEEP .01,A
30 NEXT A
```

You can modify this little program long and hard to get newer and newer effects. Try to modify the limits, step, cycle direction, and duration in the VER statement. I think everyone knows how to do this, so I won't give you examples. During your experiments, remember that low notes will not play at low durations, and the lower the note, the longer the duration should be.

When you get bored with the activity suggested in the previous paragraph, you can move on to the next step - changing the "guts" of the cycle...

It is possible to increase the number of VEER operators. Moreover, they can be set to the same frequencies and durations as well as to different ones:

```
10 FOR A=0 TO 20
20 BEEP .007,A+7: BEEP .003,A
30 NEXT A
```

The sound range of additional VEERs can be limited:

```
10 FOR A=0 TO 69
20 BEEP .01,A
30 IF A>20 AND A<40 THEN BEEP .01,A+8
40 NEXT A
```

Or like this:

```
10 FOR A=0 TO 69
20 BEEP .0003,A-6: BEEP .001,A-3: BEEP .01,A
30 NEXT A
```

The last effect is interesting in that in it the second and third VEERS enter gradually (see footnote on previous page).

The effect can be slowed down with the PAUSE operator:

```
10 FOR A=10 TO 40 STEP 2
20 BEEP .01,A: PAUSE 2
30 NEXT A
```

You can change the duration during playback:

```
10 LET T=.01
20 FOR A=10 TO 60: BEEP T,A
30 LET T=T-.0002: NEXT A
```

```
10 FOR A=0 TO 60
20 BEEP A/500,60-A
30 NEXT A
```

In the first of these examples, try changing the step or direction of the duration, its initial value.

The next variant is nested loops, of which there can be any number of loops. VEERS can occur in any of them:

```
10 FOR A=2 TO 5: BEEP .01,A
20 FOR B=20 TO 30: BEEP .01,B
30 NEXT B: NEXT A
```

```
10 FOR A=1 TO 40 STEP 10
20 FOR B=A TO A+20
30 BEEP .03,B
40 NEXT B: NEXT A
```

Try combining several cycles. For example:

```
10 FOR A=12 TO 69: BEEP .001,A
20 NEXT A
30 FOR A=69 TO 12 STEP -1
40 BEEP.001,A: NEXTA
```

Finally, an element of randomness can be added to this whole collection:

```
10 FOR A=1 TO 10: LET F=RND*50
20 BEEP .1,F: BEEP .1,F+7: BEEP .1,F+4
30 NEXT A
```

```
10 FOR A=1 TO 20
20 BEEP .1,RND*20+40
30 NEXT A
```

```
10 FOR A=1 TO 20
20 BEEP RND/6,RND*60
30 NEXT A
```

All of the above methods of creating effects can be used together, in any combination.

Search, try, and you will find among the many effects that are suitable for your program.

2.2. Making music on VEER.

Unfortunately, it is impossible to create complex music on the VEER operator, but it is possible to play

a simple tune can be played on it.

The first thing that comes to mind when trying to create a tune is to make a chain of VEERs, which promises to be of very uncertain dimensions. This method is primitive, inconvenient, space-consuming, and generally not very pretty. There are several worthy substitutes for it. For example, you can use the following loop:

```
10 FOR A=1 TO 10
20 READ T,F: BEEP T,F
30 NEXT A
```

And somewhere in the program, insert a line that starts with the DATA operator and contains the durations and pitches of the melody notes. In this example there should be ten of each.

There are pauses in some pieces of music. To play such tunes, you'll need to modify the above cycle a bit:

```
10 FOR A=1 TO 10: READ T,F
20 IF T=0 THEN PAUSE F*50: NEXT A
30 BEEP T,F: NEXT A
```

Now all you have to do is insert 0 in the data, followed by the pause duration in seconds. You may have noticed that the pause value is multiplied by 50. This is because the parameter of the PAUSE operator is counted in fiftieths of a second.

Table 1 shows how notes correspond to the durations of the VER and PAUSE operators. This table gives the standard values of notes and pauses, but in principle you can use any numbers.

For example, here is a program that sings a verse of the children's song "A Christmas tree was born in a forest:

```
10 FOR A=1 TO 29: READ T,F
20 IF T=0 THEN PAUSE F*50: NEXT A
30 BEEP T,F: NEXT A
9000 DATA .25,7.02,.25,15.86,.25,15.86,.25,14.039,.25,15.86
9010 DATA .25,12,.25,7.02,.25,7.02,.25,7.02,.25,15.86,.25,15.86
9020 DATA .25,14.039,.25,20.84,.5,19.02,0,.5,.25,19.02,.25,8.84
9030 DATA .25,8.84,.25,16.98,.25,16.98,.25,15.86,.25,14.039,.25,12
9040 DATA .25,7.02,.25,15.86,.25,15.86,.25,14.039,.25,15.86,.5,12
```

In this program it is possible to introduce the possibility of changing the tempo. To do this, replace the first three lines with the following:

```
5 LET N=.5
10 FOR A=1 TO 29: READ T,F
20 IF T=0 THEN PAUSE F*N*50: NEXT A
30 BEEP T*N,F: NEXT A
```

Now you can achieve the desired tempo of the melody by inserting different values into the fifth line. However, I do not advise you to insert 0 and very large numbers there.

You can still improve this program. For example, you can insert the keypress control, make it more flexible with respect to the length of the melody, make it a subprogram, etc. The final variant may look like this:

```
9090 DATA 100
9100 LET N=.5: RESTORE 9000
9110 READ T,F
9120 IF T=100 OR INKEY$<>"" THEN RETURN
9130 IF T=0 THEN PAUSE F*N*50: GO TO 9110
9140 BEEP T*N,F: GO TO 9110
```

Delete all lines 5 through 30 of the previous program and enter these. You can now call the melody with GO SUB 9100. Besides, you don't have to calculate the number of notes in the melody: just insert number 100 at the end of the melody (which is actually done in line 9090). You can also interrupt the performance at any time by pressing any key. If the melody ends, the subroutine will also end its work. When you restart the melody will start again, but if you remove the RESTORE operator 9000 it will continue from where it was interrupted (unless of course you used the READ or RESTORE operator after it was called).

It is possible to make playback cyclic. To do this you would have to change a couple more lines:

```
9120 IF INKEY$<>"" THEN RETURN
9125 IF T=100 THEN GO TO 9100
```

Now you can return from the subprogram only by pressing a key. But the melody will sound until someone is completely bored.

Programs of this type are ideal for simple performance, without replacing or saving the melody, but if you want to record your work on a tape recorder and load it directly from a running program, the following fragment will work for you:

```
10 FOR A=3 TO D(1)*2+2 STEP 2
20 IF D(A)=0 THEN PAUSE D(A+1)*D(2)*50: NEXT A
30 BEEP D(A)*D(2),D(A+1)
40 NEXT A
```

To make it work, you must declare and fill the array D, containing the following information about the melody: D(1) - number of notes

D(2) - rate of execution

D(3) - duration 1

D(4) - note 1

.

.

.

D(N) - duration N

D(N+1) -note N

It can be loaded from the cassette by giving a direct command

```
LOAD "name" DATA D()
```

or fill in from the DATA lines with the following program:

```
10 DIM D(20): RESTORE
20 FOR A=1 TO 20: READ D(A)
30 NEXT A
```

Note that the first two digits of the data should be the number of notes and the tempo.

The array created in this way can be saved on the tape with the command

```
SAVE "name" DATA D()
```

Here is another example of such a program, designed according to the latest fashion:

```
9100 LET A=2
9110 IF D(A)=100 THEN GO TO 9100
9120 IF INKEY$<>"" THEN RETURN
9130 IF D(A)=0 THEN PAUSE D(A+1)*D(1)*50: GO TO 9150
9140 BEEP D(A)*D(1),D(A+1)
9150 LET A=A+2: GO TO 9110
```

The array for this subprogram should be: D(1) - execution rate

D(2) - duration 1

D(3) - note 1

.

.

.

D(N) - duration N

D(N+1) -note N

D(N+2) - 100

Note that you don't need to specify the number of notes. You can even make a music editor based on this example.

Now let's go back to making data chains.

Suppose you see a note exactly like the one in Fig. 4. This is the C of the first octave. From Figure 3, you can see that its frequency is written as 8.84.

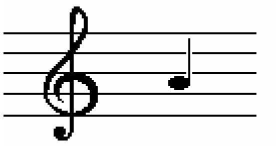


Рис. 4

This is one fourth, and looking at Table `1,` we can determine that its duration would be `.25.` Thus, the data pair for this note is `.25,8.84.` The calculations for all subsequent notes will be the same.

Нота		Зн. BEEP	Зн. PAUSE
Одна шестнадцатая	♪	.0625	3.125
Одна шестнадцатая с точкой	♪.	.09375	4.6875
Одна восьмая	♪	.125	6.25
Одна восьмая с точкой	♪.	.1875	9.375
Одна четвертая	♩	.25	12.5
Одна четвертая с точкой	♩.	.375	18.75
Одна вторая	𝅗𝅥	.5	25
Одна вторая с точкой	𝅗𝅥.	.75	37.5
Целая	𝅝	1.0	50

Табл. 1. Длительности нот для BEEP и PAUSE.

3. How the sound comes out.

Unfortunately, the ZX-Spectrum has only one bit dedicated to sound control. It is bit D4 of port 254 (#FE). But in spite of this, the programmers manage to create quite beautiful melodies and effects.

When this bit is set, the speaker (amplifier) is energized and its diaphragm is in one position. When it is reset, the speaker is not energized and the diaphragm is in a different position. Therefore, if you change the state of this bit at a high enough frequency, the diaphragm will vibrate and you will hear sound.

You can even use this method from BASIC:

```
10 FOR A=1 TO 300: OUT 254,23
20 OUT 254.7: NEXT A
```

And just as easily in assembler, but in this case you have to disable interrupts if you want to get quality sound:

```
10          DI                   ; interruption ban
20          LD    BC,2560        ; BC=duration
30          LD    A,7            ; A=color of the border
40 BEGIN    XOR   16             ; inverting the D4 bit
50          OUT   (254),A        ; output A to port 254
60          LD    D,100          ; D=delay (frequency)
70 PAUSE    DEC   D              ; D=D-1
80          JR    NZ,PAUSE       ; If D<>0, the loop
90          DEC   BC             ; BC=BC-1
100         LD    D,A            ; preservation A
110         LD    A,B            ; BC=
120         OR    C              ;      0 ?
130         LD    A,D            ; Recovery A
140         JR    NZ,BEGIN       ; If BC<>0, the loop
150         EI                   ; interrupt resolution
160         RET                  ; back to BASIC
```

You may have noticed that in these programs the port outputs not 16 and 0, which would correspond to setting and resetting bit D4, but 23 and 7. The fact is that this port besides the speaker also controls the color of the curb and the output to the recorder. Let's get a closer look at its capabilities:

bits D0 ... D2 define the color of the border:

000(0) - black   100(4) - green
001(1) - blue   101(5)-blue
010(2) - red   110(6) - yellow
011(3) - purple111(7) - white bit

D3 controls the output to the recorder, bit D4 controls the sound,
bits D5 ... D7 are not used.

When a byte is input from the port, bit D6 controls the recorder input.

I think it is now clear where the numbers 23 and 7 come from: To set and reset bit D4 you must sequentially put values 16 and 0 into the port. But you must keep the color of the border equal to 7 (white). Therefore, the first value will be 16+7=23 and the second one will be 0+7=7. Actually, you can set any color of the border. Moreover, you can create different color effects on it. To do this, just use different colors of the border when inverting the sound bit.

You may have noticed that in the above assembly language program, interrupts are forbidden. I think it is worth explaining this. The thing is that ZX Spectrum is built in such a way, that every fiftieth of a second ROM subroutine located at address 56 (#38) is called. If you want to get a quality (not crackling) sound signal, you must disable it, which is achieved by disabling interrupts (DI command). When you return to the BASIC, the interrupts must be enabled again (EI command).

4. Sound Programming in Codes.

The simplest example of playing a sound signal in codes was given in the previous chapter. In principle, this is the only way to get the sound, but it can be designed in different ways. For example, as a universal subroutine:

```
10          DI                  ; interruption ban
20 BEEP     LD    A,B           ; A=color of the border
30          SET   4,A           ; bit D4=1
40          OUT   (254),A       ; output A to port 254
50          PUSH  HL            ; HL preservation
60 LOOP1    DEC   HL            ; HL=HL-1
70          LD    A,H           ; HL=
80          OR    L             ; 0 ?
90          JR    NZ,LOOP1      ; If not, the cycle
100         POP   HL            ; HL reconstruction
110         LD    A,C           ; A= effect color
120         OUT   (254),A       ; output A to port 254
130         PUSH  HL            ; HL preservation
140 LOOP2   DEC   HL            ; HL=HL-1
150         LD    A,H           ; HL=
160         OR    L             ; 0 ?
170         JR    NZ,LOOP2      ; If not, the cycle
180         POP   HL            ; HL reconstruction
190         DEC   DE            ; DE=DE-1
200         LD    A,D           ; DE=
210         OR    E             ; 0 ?
220         JR    NZ,BEEP       ; If not, the cycle
230         EI                  ; interrupt resolution
240         RET                 ; back to BASIC
```

Before calling this subroutine you must put the frequency in register pair HL, the duration in DE, the color of the border in B, and the effect color in C (if you want the border to be monochromatic, C must be B). In addition, if you add 8 to register B (set bit D3), the signal will be fed to the recorder together with the speaker. You can make this program universal in a slightly different sense:

```
10          DI                  ; interruption ban
20          LD    A,(23624)     ; A=
30          SRL   A             ;   color
40          SRL   A             ;   bor-
50          SRL   A             ;   dura
60 BEEP     XOR   16            ; inverting the D4 bit
70          OUT   (254),A       ; output A to port 254
80          LD    C,A           ; preservation A
90          PUSH  HL            ; HL preservation
100 PAUSE   DEC   HL            ; HL=HL-1
110         LD    A,H           ; HL=
120         OR    L             ; 0 ?
130         JR    NZ,PAUSE      ; If not, the cycle
140         POP   HL            ; HL reconstruction
150         DEC   DE            ; DE=DE-1
160         LD    A,D           ; DE=
170         OR    E             ; 0 ?
180         LD    A,C           ; Recovery A
190         JR    NZ,BEEP       ; if DE<>0, the loop
200         EI                  ; interrupt resolution
210         RET                 ; back to BASIC
```

This subroutine only needs to pass the frequency in the HL register and the duration in DE, and the color of the border remains the same as it was when it was called (unless it was set by the OUT operator).

Approximately the same subroutine is available in the ZX-Spectrum ROM. It is located at address 949 (#03B5). Parameters, as before, are passed in registers HL and DE. Their values can be calculated by the following formulas:

HL=INT(437500/f-30.125+.5) DE=INT(f*t+.5)

- where f is the frequency in Hz, at- time in sec.

The frequency of notes for the fifth octave can be taken from Table 2. To raise or lower a note by an octave, its frequency must be multiplied or divided by 2. Moreover, the values obtained by division are slightly more accurate than by multiplication.

Нота	Обозначение	Частота, Гц	Октава
ДО	C	4186.01	5
ДО-диез	C#	4434.92	
РЕ	D	4698.64	
РЕ-диез	D#	4978.03	
МИ	E	5274.04	
ФА	F	5587.65	
ФА-диез	F#	5919.91	
СОЛЬ	G	6271.93	
СОЛЬ-диез	G#	6644.87	
ЛЯ	A	7040.00	
ЛЯ-диез	A#	7458.62	
СИ	B	7902.13	
ДО	C	8372.02	6

Табл. 2. Частоты нот для 5-ой октавы.

When generating a sound with all the methods above, keep in mind that the higher the tone, the shorter the duration will be.

If you want to use the familiar parameters of the VEER statement when programming sound in codes, you can use the ROM subroutine located at address 1016 (#03F8). However, there are some difficulties associated with it.

The point is that parameters are passed to it through the calculator stack, and it's not so easy to put a fractional or negative number there.

To get around this inconvenience there are three main ways. The first is to store the necessary values in standard five-byte form and put them on the stack using a special ROM subroutine. The second is to store these values in symbolic form and put them on the stack with another special ROM subroutine. And the third is to change the parameter system a little bit and do the calculations.

The first method is not good because of too much memory consumption. The second method is not good for the same reason, plus it is too slow. That leaves the third way. It is quite suitable and easy to implement.

Let the durations be given in hundredths of a second, and the note pitches, as before, in semitones above or below BEFORE the first octave. This system of parameters makes it possible to obtain notes of up to 2.55 seconds duration with frequencies ranging from -60 to 69 (like in BASIC). The duration of the note will be set in the C register, and the pitch in the B register.

Here is the subroutine that does all of the above:

```
10          PUSH  BC          ; saving BC
20          LD    A,C         ; A=C (duration)
30          CALL  11560       ; put A on the stack of the calculator
40          LD    A,100       ; A=100
50          CALL  11560       ; put A on the stack of the calculator
60          RST   40          ; calculator call
70          DEFB  5,56        ; dividing the duration by 100
80          POP   BC          ; BC recovery
90          LD    A,B         ; A=B (frequency)
100         BIT   7,A         ; A - negative ?
110         JR    NZ,MINUS    ; if so, switch to MINUS
120         CALL  11560       ; otherwise, put A on the stack of the
                                calculator
130         JP    1016        ; calling a playback subroutine
140 MINUS   NEG               ; in A - absolute value
150         CALL  11560       ; put A on the stack of the calculator
160         RST   40          ; calculator call
170         DEFB  27,56       ; sign change
```

If your assembler does not "digest" negative numbers (there are such "things"), the necessary values can be calculated using the following formula: n=256-x, - where x is the absolute value of the value and n is the result.

4.1. Programming sound effects.

Generally speaking, programming effects in code is not much different from doing it in Basic, but you get a significant advantage at the expense of speed. Therefore, in addition to pure tone, you can also create noise in assembler.

4.1.1. Tone.

Let's start with the effects based on tone generation. What are they? About the same as BASIC effects: a sound with a smoothly varying frequency. You can even use the same subroutine from ROM to make it sound alike:

```
10          LD    B,30        ; B=number of notes
20          LD    HL,100      ; HL= initial frequency
30 LOOP     LD    DE,2        ; DE=duration
40          PUSH  HL          ; HL preservation
50          PUSH  BC          ; saving BC
60          CALL  949         ; ROM subroutine call
70          POP   BC          ; BC recovery
80          POP   HL          ; HL reconstruction
90          LD    DE,25       ; DE=frequency step
100         ADD   HL,DE       ; HL increase
110         DJNZ  LOOP        ; cycle
120         RET               ; back to BASIC
```

Most likely, you noticed that interrupts are not forbidden in this effect, and the signal quality is not degraded at all (see the chapter "How the sound is produced"). The point is that the called subroutine does everything necessary by itself.

If you have already managed to type and run this fragment, you probably felt the difference in sound, and not in favor of Basic.

It is possible to modify this effect by changing not the whole frequency, but only its low-order byte:

```
10          LD    B,60        ; B=number of notes
20          LD    C,50        ; C=frequency step
30          LD    HL,300      ; HL= initial frequency
40 LOOP     LD    DE,10       ; DE=duration
50          PUSH  HL          ; HL preservation
60          PUSH  BC          ; saving BC
70          CALL  949         ; ROM subroutine call
80          POP   BC          ; BC recovery
90          POP   HL          ; HL reconstruction
100         LD    A,L         ; increased-
110         ADD   C           ;   .
120         LD    L,A         ; L
130         DJNZ  LOOP        ; cycle
140         RET               ; back to BASIC
```

From using ROM subroutines, let's move on to creating our own. The sound of the previous examples can be made smoother and more extended:

```
10          DI                ; interruption ban
20          XOR   A           ; A= border color (0)
30          LD    B,255       ; B=initial frequency
40          LD    C,255       ; C=duration
50 BEEP     XOR   16          ; inverting the D4 bit
60          OUT   (254),A     ; output A to port 254
70          PUSH  BC          ; saving BC
80 LOOP     DJNZ  LOOP        ; delay
90          POP   BC          ; BC recovery
100         DEC   B           ; B reduction
110         DEC   C           ; C=C-1
120         JR    NZ,BEEP     ; If C<>0, the loop
```

```
130          EI                  interrupt resolution
140          RET               ; back to BASIC
```

The possibilities of this effect are easily increased:

```
10           DI                ; interrupt denial
20           XOR   A           ; A= border color (0)
30           LD    B,255       ; B=number of notes
40           LD    C,1         ; C=initial frequency
50  LOOP1    PUSH  BC          ; saving BC
60           LD    B,5         ; B=duration
70  LOOP2    XOR   16          ; inverting the D4 bit
80           OUT   (254),A     ; output A to port 254
90           PUSH  BC          ; saving BC
100          LD    B,C         ; B=C
110  LOOP3   DJNZ  LOOP3       ; delay
120          POP   BC          ; BC recovery
130          DJNZ  LOOP2       ; cycle
140          POP   BC          ; BC recovery
150          INC   C           ; increase C
160          DJNZ  LOOP1       ; second round
170          EI                ; interrupt resolution
180          RET               ; back to BASIC
```

Most likely, you noticed that in the first variant the delay decreases, and in the second variant it increases. Nothing prevents you from changing its direction. To do this, just select the command DEC (decrease) or INC (increase). You can also adjust the duration, start frequency, etc.

You can change the frequency offset step:

```
10           DI                ; interruption ban
20           XOR   A           ; A= border color (0)
30           LD    B,255       ; B=initial frequency
40           LD    C,255       ; C=duration
50  BEEP     XOR   16          ; inverting the D4 bit
60           OUT   (254),A     ; output A to port 254
70           PUSH  BC          ; saving BC
80  LOOP     DJNZ  LOOP        ; delay
90           POP   BC          ; BC recovery
100          EX    AF,AF'      ; change the A and F registers to
                                 alternate registers
110          LD    A,B         ; A=B
120          SUB   3           ; A=A-3
130          LD    B,A         ; B=A
140          EX    AF,AF'      ; reverse register change
150          DEC   C           ; C=C-1
160          JR    NZ,BEEP     ; If C<>0, the loop
170          EI                ; interrupt resolution
180          RET               ; back to BASIC
```

Now let's take advantage of the leeway given to us by programming in codes. Let's write a non-standard playback procedure:

```
10           DI                ; interruption ban
20           LD    D,10        ; D=delay 1
30           LD    E,100       ; E=delay 2
40           LD    C,255       ; C=duration
50           XOR   A           ; A= border color (0)
60  LOOP1    XOR   16          ; inverting the D4 bit
70           OUT   (254),A     ; output A to port 254
80           LD    B,D         ; B=D
90  LOOP2    DJNZ  LOOP2       ; delay
100          XOR   16          ; inverting the D4 bit
110          OUT   (254),A     ; output A to port 254
120          LD    B,E         ; B=E
130  LOOP3   DJNZ  LOOP3       ; delay
140          INC   D           ; increase   D
150          INC   E           ; increase   E
160          DEC   C           ; C=C-1
170          JR    NZ,LOOP1    ; if C<>0,   then the
                                            cycle
```

```
180          EI                  ; interrupt resolution
190          RET                 ; return to BASIC
```

Here, too, you can vary the parameters long and painstakingly (see Appendix 1 - Flowing 2).

Quite an interesting effect is obtained when using the R register, the value of the lower seven bits increases after the next machine cycle is executed. This effect can be conventionally called "half-noise".

Here is an example of this effect:

```
10           DI                  ; interruption ban
20           LD    C,53          ; C=delay 1
30           LD    B,207         ; B=delay 2
40           LD    E,203         ; E=duration
50           LD    D,0           ; D=color of the curb
60           LD    A,128         ; A=tempo (0/128)
70           LD    R,A           ; R=A
80 BEGIN     LD    A,R           ; A=R
90 PAUS1     DEC   A             ; A=A-1
100          JR    NZ,PAUS1      ; If A<>0, the loop
110          LD    A,D           ; A=D
120          OR    16            ; setting bit D4
130          OUT   (254),A       ; output A to port 254
140          LD    A,C           ; A=C
150 PAUS2    DEC   A             ; A=A-1
160          JR    NZ,PAUS2      ; If A<>0, the loop
170          LD    A,D           ; A=D
180          OUT   (254),A       ; output A to port 254
190          LD    A,B           ; A=B
200 PAUS3    DEC   A             ; A=A-1
210          JR    NZ,PAUS3      ; If A<>0, the loop
220          INC   C             ; C=C+1
230          INC   B             ; B=B+1
240          DEC   E             ; E=E-1
250          JR    NZ,BEGIN      ; if E<>0, the loop
260          EI                  ; interrupt resolution
270          RET                 ; Returns
```

The values on lines 20 and 30 define the delays between level drops, line 40 the duration of the effect, line 50 the color of the border, and line 60 the tempo. In line 60 it makes sense to use only the values 0 and 128, as all the others will be similar to these. In lines 220 and 230 you can set the law of change of both delays (INC, DEC and NOP commands with registers B and C in any combination).

If you're not confused by all these examples, let's move on to generating noise. If you still don't understand anything, I suggest that you try these effects and change their parameters.

4.1.2. Noise.

What is noise? It is a sequence of pulses of random duration. That's why we need random data to create it. Where do we get it? The only good source is the ROM where the BASIC interpreter is stored. The ROM is from address 0 to 16383 (#3FFF). However, these values will be not quite random, or rather not random at all, but they will suit us.

You can create noise in two different ways. They are slightly different in sound. The first is to output values read from the ROM to the port. The second is to use these values as a delay.

When generating noise by any of these methods it is not necessary to disable interruptions, because crackling will not be heard in the noise.

With the first method, you can extract data from each byte for eight passes of the playback cycle, but this is not advantageous from a programmatic point of view. Therefore, most effects only take one value from a byte. For example:

```
10           LD    HL,0          ; HL = ROM address
20           LD    BC,1000       ; BC=duration
```

```
30 BEGIN    PUSH  BC          ; saving BC
40          LD    A,(HL)      ; A=cell content      ROM
50          AND   240         ; reset the curb bits
60          OUT   (254),A     ; output A to port
                                254
70          LD    B,50        ; B=frequency
80 LOOP     DJNZ  LOOP        ; delay
90          INC   HL          ; HL=HL+1
100         POP   BC          ; BC recovery
110         DEC   BC          ; BC=BC-1
120         LD    A,B         ; BC=
130         OR    C           ; 0 ?
140         JR    NZ,BEGIN    ; cycle
150         RET               ; back to BASIC
```

An example of the second method:

```
10          LD    HL,0        ; HL = ROM address
20          LD    BC,1000     ; BC=duration
30          XOR   A           ; A= border color (0)
40 BEGIN    PUSH  BC          ; saving BC
50          XOR   16          ; bit inversion       D4
60          OUT   (254),A     ; output A to port
                                254
70          LD    B,(HL)      ; B=component of the  ROM
                                cell
80 LOOP1    DJNZ  LOOP1       ; delay
90          LD    B,40        ; B=frequency
100 LOOP2   DJNZ  LOOP2       ; delay
110         INC   HL          ; HL=HL+1
120         POP   BC          ; BC recovery
130         DEC   BC          ; BC=BC-1
140         LD    D,A         ; preservation A
150         LD    A,B         ; BC=
160         OR    C           ; 0 ?
170         LD    A,D         ; Recovery A
180         JR    NZ,BEGIN    ; cycle
190         RET               ; back to BASIC
```

If you change the `INC HL` command to `INC L` in these effects, their sound will become jumpy. This is because the data is not taken from the entire given amount of ROM, but from a 256-byte part of it. And when this part ends, the reading continues from the beginning.

Try changing the other parameters.

The next step is noise with a varying frequency. It will look something like this:

```
10          LD    HL,0        ; HL = ROM address
20          LD    B,100       ; B=length of effect
30          LD    C,10        ; C=initial frequency
40 LOOP1    PUSH  BC          ; saving BC
50          LD    B,20        ; B=duration
60 LOOP2    LD    A,(HL)      ; A=the contents of the
                                ROM cell
70          AND   240         ; reset the curb bits
80          OUT   (254),A     ; output A to port 254
90          PUSH  BC          ; saving BC
100         LD    B,C         ; B=C
110 LOOP3   DJNZ  LOOP3       ; delay
120         INC   HL          ; HL=HL+1
130         POP   BC          ; BC recovery
140         DJNZ  LOOP2       ; cycle
150         POP   BC          ; BC recovery
160         INC   C           ; increased delay
170         DJNZ  LOOP1       ; cycle
180         RET               ; back to BASIC
```

This effect, too, can be changed beyond recognition: change the length, frequency, duration, frequency offset step, frequency offset direction (`INC C` or `DEC C` command), and the type of noise (`INC HL` or `INC L` command).

Another variation on the noise effect:

```
10          LD    HL,0            ; HL = ROM
                                    address
20          LD    D,100          ; D=delay 1
30          LD    E,10           ; E=delay 2
```

```
40          LD    C,255       ;   C=duration
50          XOR   A           ; A=color of the (0)
                                  border
60 BEGIN    XOR   16          ;   inverting   bat  D4
70          OUT   (254),A     ;  output A to  254
                                  the port
80          LD    B,(HL)      ; B=component of the  ROM
                                cell
90 LOOP1    DJNZ  LOOP1       ; delay 1
100         LD    B,D         ; B=D
110  LOOP2  DJNZ  LOOP2       ; delay 2
120         XOR   16          ; bit inversion       D4
130         OUT   (254),A     ; output A to port
                                254
140         LD    B,(HL)      ; B=component of the  ROM
                                cell
150  LOOP3  DJNZ  LOOP3       ; delay 3
160         LD    B,E         ; B=E
170  LOOP4  DJNZ  LOOP4       ; delay 4
180         INC   HL          ; HL=HL+1
190         INC   D           ; increment D
200         INC   E           ; increase E
210         DEC   C           ; C=C-1
220         JR    NZ,BEGIN    ; If C<>0, the loop
230         RET               ; back to BASIC
```

You can mock this example as long as you mock all the previous ones.

All of the effects in this chapter can be viewed as blanks. You may have to work a little harder to get the final variants. As already mentioned, they all have a huge number of variants. You can also combine several effects together, or call them to run in a loop, etc. All the basic principles of combining effects are listed in chapter 2.1.

All effects are written so that you can change as many parameters as possible. In cases of specific application, they can be greatly simplified. Here is an example of a combined and simplified effect:

```
10          DI                ; interruption ban
20          LD    E,100       ; E = cycle time
30          LD    C,0         ; C= border color
40          LD    B,4         ; B=number of cycles
50          LD    L,1         ; L=frequency offset
60          LD    H,30        ; H=initial frequency
70 LOOP1    LD    D,E         ; D=E
80 LOOP2    LD    A,C         ; A=C
90          XOR   16          ; inverting the D4 bit
100         OUT   (254),A     ; output A to port 254
110         LD    C,A         ; C=A
120         LD    A,H         ; A=H
130         ADD   A,L         ; add L to A
140         LD    H,A         ; H=A
150  LOOP3  DEC   A           ; A=A-1
160         JR    NZ,LOOP3    ; If A<>0, the loop
170         DEC   D           ; D=D-1
180         JR    NZ,LOOP2    ; If D<>0, the loop
190         LD    A,L         ; A=L
200         NEG               ; shift the sign of the    A
                                register
210         LD    L,A         ; L=A
220         DJNZ  LOOP1       ; cycle
230         EI                ; interrupt resolution
240         RET               ; Returns
```

You can customize all of the above effects to suit your own needs. For example, you can change the color of the border or make the signal be output to a tape recorder besides the speaker (to do that, change all XOR 16 commands to XOR 24, and AND 240 to AND 248).

4.1.3. Effect Complexes.

Usually games (as well as any other programs) use more than one or two sound effects. That is

why it is convenient to combine effects into groups (complexes) and call them for execution by one subprogram, passing the number to it as a parameter

effect.

If the effects are varied and played by different subprograms, it is best to store the addresses of these subprograms in the effects table. In this case, you can use such a subprogram to play back the effects:

```
10          ADD   A,A         ; A=A*2
20          LD    E,A         ; DE
30          LD    D,0         ; =A
40          LD    HL,TABLE    ; HL=table address
50          ADD   HL,DE       ; HL=HL+DE
60          LD    E,(HL)      ; DE=
70          INC   HL          ; address
80          LD    D,(HL)      ; subprograms
90          EX    DE,HL       ; exchange the values of HL
                                and DE
100         JP    (HL)        ; Calling an effect
                                subroutine
110 TABLE   DEFW  ...         ; address table
```

Before calling this subroutine, you must put the effect number in the A register. Don't forget to also fill in the table with the addresses of the effects. Naturally, the effects themselves must be located at the specified addresses. Under no circumstances should the effect number be greater than the number of subprograms described in the table, otherwise the computer will "freeze" or "reset". Note also that this subprogram can work with a maximum of 127 effects.

If the effects are the same type and played by the same subroutine, you can store the parameters of that subroutine in the effects table. Here is an example that uses this method:

```
10          LD    E,A         ; A
20          ADD   A,A         ;  =
30          ADD   A,E         ;   A*3
40          LD    E,A         ; DE
50          LD    D,0         ;   =A
60          LD    HL,TABLE    ; HL=table address
70          ADD   HL,DE       ; HL=HL+DE
80          LD    C,(HL)      ; C=duration
90          INC   HL          ; HL=HL+1
100         LD    E,(HL)      ; E=frequency
110         INC   HL          ; HL=HL+1
120         LD    A,(HL)      ; A=frequency change
130         LD    (CHNG),A    ; frequency change setting
140         DI                ; interruption ban
150         XOR   A           ; A= border color (0)
160 BEGIN   XOR   16          ; inverting the D4 bit
170         OUT   (254),A     ; output A to port 254
180         LD    B,E         ; B=E
190 PAUSE   DJNZ  PAUSE       ; delay
200 CHNG    NOP               ; frequency change reserve
210         DEC   C           ; C=C-1
220         JR    NZ,BEGIN    ; If C<>0, the loop
230         EI                ; interrupt resolution
240         RET               ; Returns
250 TABLE   DEFB  0,0,28      ; table
260         DEFB  0,0,29      ;
270         DEFB  0,128,28    ;   effects
280         DEFB  0,128,29    ;
```

Before calling this subroutine, you must enter an effect number in the A register. Four effects have already been created in this program, but you can change them or increase their number. If you call an effect whose number is greater than the number of the described subroutines, something awful can also happen.

In this example, three bytes are allocated to describe the effect. The first byte is the duration of the effect, the second is the initial frequency, and the third is how the frequency is changed. The third byte can take the values 0, 28 and 29. This means, respectively, to keep, increase, and decrease the frequency. Do not enter any other values in this byte, as this can have unpredictable consequences.

This program is just an example. You can customize it to work with absolutely any effect.

This subprogram can handle a maximum of 85 effects.

In both of the above subroutines, the numbering of the effects starts from zero.

4.2. Volume control.

Actually, the ZX-Spectrum is absolutely not adapted for volume control, but it is still possible to change it within small limits thanks to the relatively high clock frequency of the Z-80 microprocessor (about 3.5 MHz).

As discussed in Chapter 2, when you send a signal to a speaker, its diaphragm changes position. If you send a signal back while the diaphragm is moving, it will return to its original position without having traveled the desired path. So the amplitude (and therefore the volume) will be less than normal. For example, compare the sound of these two almost identical programs:

```
10  PROG1    DI                 ; interruption ban
20           LD    B,255        ; B=frequency
30           LD    C,255        ; C=duration
40           XOR   A            ; A= border color (0)
50  LOOP1    XOR   16           ; inverting the D4 bit
60           OUT   (254),A      ; output A to port 254
70           PUSH  BC           ; saving BC
80  LOOP2    DJNZ  LOOP2        ; delay
90           POP   BC           ; BC recovery
100          DEC   C            ; C=C-1
110          JR    NZ,LOOP1     ; If C<>0, the loop
120          EI                 ; interrupt resolution
130          RET                ; back to BASIC

10  PROG2    DI                 ;
20           LD    B,255        ;
30           LD    C,255        ;
40           XOR   A            ;
50  LOOP1    XOR   16           ; all
60           OUT   (254),A      ;
70           XOR   16           ; (except for these two
                                  lines,
80           OUT   (254),A      ; repeating the previous
                                  ones)
90           PUSH  BC           ;
100  LOOP2   DJNZ  LOOP2        ; similarly
110          POP   BC           ;
120          DEC   C            ; previous
130          JR    NZ,LOOP1     ;
140          EI                 ; for example !
150          RET                ;
```

When you type them in assembler and run them, you will be very surprised. But you can't argue against the facts - the second subroutine will sound several times quieter than the first one!

Here is the text of a more universal subprogram:

```
10           DI                 ; interruption ban
20           LD    B,50         ; B=volume
30           LD    C,50         ; C=frequency
40           LD    D,15         ; D=duration
50           LD    E,50         ; E=number of notes
60           XOR   A            ; A= border color (0)
70  LOOP1    PUSH  DE           ; DE retention
80  LOOP2    PUSH  BC           ; saving BC
90           XOR   16           ; inverting the D4 bit
100          OUT   (254),A      ; output A to port 254
110  LOOP3   DJNZ  LOOP3        ; delay (volume)
120          XOR   16           ; inverting the D4 bit
130          OUT   (254),A      ; output A to port 254
140          LD    B,C          ; B=C
150  LOOP4   DJNZ  LOOP4        ; delay (frequency)
```

```
160         POP   BC           ; BC recovery
170         DEC   D            ; D=D-1
180         JR    NZ,LOOP2     ; If D<>0, the loop
190         POP   DE           ; DE recovery
200         NOP                ; reserve
210         NOP                ; reserve
220         NOP                ; reserve
230         DEC   E            ; E=E-1
240         JR    NZ,LOOP1     ; if E<>0, the loop
250         EI                 ; interrupt resolution
260         RET                ; back to BASIC
```

In this effect you can pick up the desired volume. But don't get too carried away: at values higher than 40-50 the volume is no longer increased. It should be noted that when you change the volume during playback, the frequency also changes. Therefore, it is necessary to provide for its compensation.

Now for the three bytes of the reserve (NOP commands). They are for changing the volume (register B), frequency (register C) and duration (register D). If you replace these three NOPs with the commands DEC B (decrease volume), INC C (frequency compensation) and INC D (increase duration), you get an interesting effect with a damped volume.

4.3. Tone Control.

The ZX-Spectrum is as unsuited to tone control as it is to volume control. But, like volume, it can be controlled within small limits.

As you already know, the tone depends on the waveform of the sound. Since it is not easy to adapt a volume control subroutine to form the tone, you will only have to make do with two levels of volume when creating different waveforms. This can be done, for example, by using the following subroutine:

```
10          DI                 ; interruption ban
20 LOOP1    LD    B,8          ; B=tempo
30 LOOP2    XOR   A            ; A= border color (0)
40          RLC   E            ; scrolling the tone through the CY
                                 flag
50          JR    NC,NOSIGN    ; If the CY=0 flag, switch to NOSIGN
60          OR    16           ; setting of bit D4 of register A
70 NOSIGN   OUT   (254),A      ; output A to port 254
80          LD    D,H          ; D=delay
90 PAUSE    DEC   D            ; D=D-1
100         JR    NZ,PAUSE     ; If D<>0, the loop
110         DJNZ  LOOP2        ; ramp cycle
120         LD    A,H          ; A=H
130         ADD   A,L          ; add L to A
140         LD    H,A          ; H=A
150         DEC   C            ; C=C-1
160         JR    NZ,LOOP1     ; If C<>0, the loop
170         EI                 ; interrupt resolution
180         RET                ; Returns
```

Before calling this subroutine you must enter the note length in register C, the frequency in register H, the frequency offset in register L (negative values and zero are possible), and the timbre in register E. Note that not all E register values are meaningful. For example, with E equal to 255 or 0 you will not hear anything at all, and the tones 1 and 128 will sound exactly the same, etc.

Essentially, it only makes sense to use the following 19 tones: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 37, 43, 45, 51 и 85. All other values are audibly perceived similarly to these.

This program is just an example. You can write your own one that plays back the sound with a 16-bit tone, or with a tone pre-recorded from the tape input (see chapter 4.6).

4.4. Music Programming.

Programming music in code is a thankless task. It is better to do it with the help of some music editor. But the editor may not meet all of your needs, so it is still advisable to learn how to program in codes.

The principle is the same – sequentially read the data and call the playback procedure. In this case it is necessary to provide an interruption of the execution (most often for this purpose the pressing of a key is controlled). For example:

```
10 START    LD    HL,60000    ; HL=data address
20 NEXT     LD    E,(HL)      ; DE=
30          INC   HL          ;   length-
40          LD    D,(HL)      ;    Tel
50          INC   HL          ;     availability
60          LD    A,D         ; DE
70          CP    255         ;   =
80          JR    NZ,CONT     ;    65535
90          LD    A,E         ;      ?
100         CP    255         ; If so,
110         JR    Z,START     ; then the tune at first
120 CONT    LD    A,D         ; DE=
130         OR    E           ;    0 ?
140         JR    NZ,BEEP     ; if not, switch to BEEP
150         LD    E,(HL)      ; DE=
160         INC   HL          ;    long-
170         LD    D,(HL)      ;    availability
180         INC   HL          ;    pauses
190 PAUSE   HALT              ; abort expectation
200         DEC   DE          ; DE=DE-1
210         LD    A,D         ; DE=
220         OR    E           ;    0 ?
230         JR    NZ,PAUSE    ; If not, the cycle
240         JR    NEXT        ; switch to processing the following
                                data
250 BEEP    LD    C,(HL)      ; BC=
260         INC   HL          ;    hour-
270         LD    B,(HL)      ;    tho-
280         INC   HL          ;    ta
290         PUSH  HL          ; HL preservation
300         PUSH  BC          ; put BC on the stack
310         POP   HL          ; place the value from the stack in
                                HL
320         CALL  949         ; Calling a playback subroutine
330         CALL  654         ; keyboard control
340         POP   HL          ; HL reconstruction
350         LD    A,E         ; A=E
360         CP    255         ; if the key is pressed,
370         RET   NZ          ; then back to BASIC
380         JR    NEXT        ; switch to processing the following
                                data
```

When calling this subroutine from machine codes, note that interrupts must be allowed! Otherwise, if there is a pause in the data first, the computer will "hang".

Before using this rather scary subroutine, you have to create a data array for it (that's the fun part). In this case it must be located at 60000 (#EA60), but you can easily change its location by changing the number in the first line of this subroutine.

The data for each note should be as follows: first two bytes of duration (the first is the lowest), then two bytes of frequency. If you put two zeros instead of the duration, the next pair of bytes will be treated as a pause length. To mark the end of the tune, insert two bytes 255 in the data instead of the next duration.

Pause duration is measured in fiftieths of a second, and probably no one knows how a note is measured, although it is possible to calculate the necessary value (see Chapter 4).

Now about the disadvantages, which, frankly speaking, are plentiful. Two of them have already been mentioned above: the difficulty of composing a melody and the rather strange measurement of

duration

notes. In addition, the duration depends on the frequency. For example, if you enter `0,1,100,0,0,1,0,5,255,255,` you will not hear two different notes of the same length as you would expect.

Another disadvantage is that you can interrupt your performance only between notes. However, this can be easily fixed:

```
10 START    LD    HL,60000    ; HL=data address
20 NEXT     LD    E,(HL)      ; DE=
30          INC   HL          ;    length-
40          LD    D,(HL)      ;    Tel
50          INC   HL          ;    availability
60          LD    A,D         ; DE
70          CP    255         ;   =
80          JR    NZ,CONT     ;    65535
90          LD    A,E         ;       ?
100         CP    255         ; If so,
110         JR    Z,START     ; the tune at first
120 CONT    LD    A,D         ; DE=
130         OR    E           ;    0 ?
140         JR    NZ,BEEP     ; if not, switch to BEEP
150         LD    E,(HL)      ; DE=
160         INC   HL          ;    long-
170         LD    D,(HL)      ;    availability
180         INC   HL          ;    pauses
190 PAUSE   XOR   A           ; A=0 (control of the entire keyboard)
200         IN    A,(254)     ; keystroke
210         CPL               ; inverting A
220         AND   31          ; discard extra bits
230         RET   NZ          ; if a key is pressed, the return
240         HALT              ; abort expectation
250         DEC   DE          ; DE=DE-1
260         LD    A,D         ; DE=
270         OR    E           ;    0 ?
280         JR    NZ,PAUSE    ; If not, the cycle
290         JR    NEXT        ; switch to processing the following
                                data
300 BEEP    LD    C,(HL)      ; BC=
310         INC   HL          ;    hour-
320         LD    B,(HL)      ;    tho-
330         INC   HL          ;    ta
340         PUSH  HL          ; HL preservation
350         PUSH  BC          ; put BC on the stack
360         POP   HL          ; place the value from the stack in HL
370         CALL  PLAY        ; calling a playback subroutine
380         POP   HL          ; HL reconstruction
390         RET   NZ          ; then back to BASIC
400         JR    NEXT        ; switch to processing the following
                                data
410 PLAY    DI                ; interruption ban
420         LD    A,(23624)   ; A=
430         SRL   A           ;   color
440         SRL   A           ;   bor-
450         SRL   A           ;   dura
460 LOOP1   XOR   16          ; inverting the D4 bit
470         OUT   (254),A     ; output A to port 254
480         LD    C,A         ; preservation A
490         PUSH  HL          ; HL preservation
500 LOOP2   XOR   A           ; A=0 (control of the entire keyboard)
510         IN    A,(254)     ; keystroke
520         CPL               ; inverting A
530         AND   31          ; discard extra bits
540         JR    Z,CONT2     ; if the key is not pressed, continue
550         POP   HL          ; clear the value from the stack
560         EI                ; interrupt resolution
570         RET               ; Returns
580 CONT2   DEC   HL          ; HL=HL-1
590         LD    A,H         ; HL=
```

```
600         OR    L          ;    0 ?
```

```
610         JR    NZ,LOOP2    ; If not, the cycle
620         POP   HL          ; HL reconstruction
630         DEC   DE          ; DE=DE-1
640         LD    A,D         ; DE=
650         OR    E           ;    0 ?
660         LD    A,C         ; Recovery A
670         JR    NZ,LOOP1    ; If DE<>0, the loop
680         EI                ; interrupt resolution
690         RET               ; Returns
```

Note that the same data will be reproduced differently in this and the previous subroutines.

Finally, the most obvious drawback is that the music produced by these subroutines is too primitive. Something more serious can be done in assembler.

I hope I convinced you not to try to program music directly in codes, but instead to get a good music editor.

All of the subroutines in this chapter can also be used to create effects. To do this, just encode the individual steps of the effect instead of the notes. Preferably, each step should be as short as possible to make the effect sound smooth.

4.5. Multi-voiced melodies.

Although in the previous chapter I tried hard to prove to you the futility of trying to program music in assembler, you should still know the basic principles of this activity.

I will now describe the creation of multivoiced music, that is, music in which two or more notes can be played at the same time.

The first example would be a subroutine that simulates two-voice sound:

```
10          DI                ; interruption ban
20          LD    D,200       ; D=frequency 1
30          LD    E,50        ; E=frequency 2
40          LD    H,150       ; H=duration
50          XOR   A           ; A=voice board 1       (0)
60          EX    AF,AF'      ; change registers A    on alternatives
                                and F
70          XOR   A           ; A=voice board 2       (0)
80          LD    C,E         ; C=counter 1
90          LD    B,D         ; B=counter 2
100 BEEP    EX    AF,AF'      ; change registers A    (voice change)
                                and F
110         DEC   C           ; C=C-1
120         JR    NZ,CONT     ; If C<>0, change to CONT
130         LD    C,E         ; Restore counter 1
140         XOR   16          ; Invert bit D4 voice 1
150 CONT    OUT   (254),A     ; output A to port 254
160         EX    AF,AF'      ; changing A and F registers (voice
                                change)
170         DEC   B           ; B=B-1
180         JR    NZ,CONT2    ; if B<>0, then go to CONT2
190         LD    B,D         ; Restore counter 2
200         XOR   16          ; Invert bit D4 of voice 2
210 CONT2   OUT   (254),A     ; output A to port 254
220         INC   L           ; L=L+1
230         JR    NZ,BEEP     ; if L<>0, switch to BEEP
240         NOP               ; reserve
250         NOP               ; reserve
260         DEC   H           ; H=H-1
270         JR    NZ,BEEP     ; if H<>0, switch to BEEP
280         EI                ; interrupt resolution
290         RET               ; Returns
```

Now let me explain some of the details.

The two alternate A registers are used by the first and second voices to store the border color and speaker state.

Strange operation c register L before ending cycle is intended for "stretching" the duration. If it is removed, the sound of this subprogram will become so

You may not even be able to hear it.

You can use the two reserve bytes (`NOP` operations) to change the frequency of both voices during playback. To do this, insert `INC D` or `DEC D` for the first voice and `INC E` or `DEC E` for the second voice instead of "`NOP`".

This subroutine can be used in assembler (see the previous chapter) or in BASIC (in this case the parameters should be entered by ROKE operator in the following cells: `ADDR+2` – frequency 1, `ADDR+4` – frequency 2, `ADDR+6` – duration, where `ADDR` – address of subroutine location in memory).

The following two multi-voiced subprograms are very similar to each other, but the first sounds much quieter and somewhat cleaner than the second.

Here they are:

```
10          DI                  ; interruption ban
20          LD    HL,2000       ; HL=duration
30  LOOP1   LD    IX,FRQS       ; IX=frequency table address
40          LD    B,5           ; B=number of votes
50  LOOP2   DEC   (IX+0)        ; byte on address IX+0 reduce by 1
60          JR    NZ,NOSIGN     ; if not 0, then switch to NOSIGN
70          LD    A,24          ; A= border color + 24
80          OUT   (254),A       ; output A to port 254
90          LD    A,(IX+5)      ; counter update
100         LD    (IX+0),A      ;   current channel
110         XOR   A             ; A= border color (0)
120         OUT   (254),A       ; output A to port 254
130 NOSIGN  INC   IX            ; IX=IX+1
140         DJNZ  LOOP2         ; cycle
150         DEC   HL            ; HL=HL-1
160         LD    A,H           ; HL=
170         OR    L             ;    0?
180         JR    NZ,LOOP1      ; If not, the cycle
190         EI                  ; interrupt resolution
200         RET                 ; Returns
210 FRQS    DEFB  1,1,1,1,1     ; Counters for 5 channels
220         DEFB  10,20,30      ; frequencies for
230         DEFB  40,50         ;    5 channels
```

```
10          DI                  ; interruption ban
20          LD    HL,2000       ; HL=duration
30  LOOP1   LD    IX,FRQS       ; IX=frequency table address
40          LD    B,5           ; B=number of votes
50  LOOP2   DEC   (IX+0)        ; byte on address IX+0 reduce by 1
60          JR    NZ,NOSIGN     ; if not 0, then switch to NOSIGN
70          LD    A,(IX+5)      ; A=content IX+5
80          XOR   1             ; inverting the D0 bit
90          LD    (IX+5),A      ; the contents of IX+5=A
100         LD    A,0           ; A=color of the border
110         JR    Z,NOSIGN      ; If D0 contains. IX+5=0,then go to   NOSIGN
120         OR    24            ; set bits D3 and D4 of register A to 1
130         OUT   (254),A       ; output A to port 254
140         LD    A,(IX+10)     ; counter update
150         LD    (IX+0),A      ;   current channel
160 NOSIGN  INC   IX            ; IX=IX+1
170         DJNZ  LOOP2         ; cycle
180         DEC   HL            ; HL=HL-1
190         LD    A,H           ; HL=
200         OR    L             ;    0?
210         JR    NZ,LOOP1      ; If not, the cycle
220         EI                  ; interrupt resolution
230         RET                 ; Returns
240 FRQS    DEFB  1,1,1,1,1     ; counters for 5 channels
250         DEFB  0,0,0,0,0     ; buffers for 5 channels
260         DEFB  10,20,30      ; frequencies for
270         DEFB  40,50         ;    5 channels
```

These subroutines set the counters for each of the channels and decrease them sequentially. If any of the counters become 0, a signal is output to the speaker, and the frequency from the table is entered into that counter.

In these examples, the subroutines play five voices each, but you can easily change their number by modifying the values in lines 40 and 90 for the first subroutine or 40, 70, 90 and 140 (line 140 must be inserted for the number of voices multiplied by 2) for the second. In this case, you should also adjust the frequency tables.

Although there can be quite a lot of voices (up to 85), you should not get carried away with their number, because the more of them, the worse the quality. However, it is unlikely that there is a genius capable of writing a melody with 85 voices.

If you want to use polyphony, but at a particular point in time you need to play fewer voices than specified, the unused voices must be assigned frequencies that coincide with the frequencies of the voices being used.

4.6. Processing of external signals.

ZX-Spectrum provides programmers with a rather interesting possibility – to read and store in memory the sound signals fed to its tape input. These fragments can be played back at different speeds, which is undoubtedly very interesting. However, the quality of the sound obtained in this way leaves much to be desired, and its maximum duration is limited to a few tens of seconds (simply running out of 48K RAM!).

But let's move on from words to deeds. Here is the subroutine that writes the sound to memory:

```
10          DI                  ; interruption ban
20          LD    HL,25000     ; HL=save audio address
30          LD    DE,35000     ; DE=duration of sound
40 LOOP1    LD    B,8          ; B= bit counter
50 LOOP2    SLA   (HL)         ; scrolling the value in memory
60          IN    A,(254)      ; enter a value from port 254
70          BIT   6,A          ; check the tape recorder bit
80          JR    Z,NOSIGN     ; if 0, then switch to NOSIGN
90          SET   0,(HL)       ; set bit D0 in memory
100 NOSIGN  LD    C,3          ; C=delay
110 PAUSE   DEC   C            ; C=C-1
120         JR    NZ,PAUSE     ; If C<>0, the loop
130         DJNZ  LOOP2        ; continue the byte cycle
140         INC   HL           ; HL=HL+1
150         DEC   DE           ; DE=DE-1
160         LD    A,D          ; DE=
170         OR    E            ;    0?
180         JR    NZ,LOOP1     ; If not, the cycle
190         EI                 ; interrupt resolution
200         RET                ; Returns
```

And this is a playback subroutine:

```
10          DI                  ; interruption ban
20          LD    HL,25000     ; HL=saved audio address
30          LD    DE,35000     ; DE=duration of sound
40          NOP                ; reserve
50          XOR   A            ; A= border color (0)
60 LOOP3    LD    B,8          ; B= bit counter
70 LOOP4    AND   239          ; reset of bit D4 of register A
80          RLC   (HL)         ; scrolling data through the CY flag
90          JR    NC,NOSGN     ; If the CY=0 flag, switch to NOSGN
100         OR    16           ; setting of bit D4 of register A
110 NOSGN   OUT   (254),A      ; output A to port 254
120         LD    C,3          ; C=delay
130 PAUS2   DEC   C            ; C=C-1
140         JR    NZ,PAUS2     ; If C<>0, the loop
150         DJNZ  LOOP4        ; continue the byte cycle
160         INC   HL           ; HL=HL+1
170         LD    C,A          ; preservation A
```

```
180          DEC    DE          ; DE=DE-1
190          LD     A,D         ; DE=
200          OR     E           ;    0?
210          LD     A,C         ; Recovery A
220          JR     NZ,LOOP3    ; if DE<>0, the loop
230          EI                 ; interrupt resolution
240          RET                ; Returns
```

The data addresses and lengths must be the same in these two subroutines. Note also that they use 35Kb of memory from address `25000`. Therefore, there should not be any data in this area of the RAM. The subroutines themselves should also be located somewhere else. The optimal way is to renumber the lines of the second subroutine, starting from `210` in steps of `10` (if you work in GENS4, enter command `N210,10` for this purpose); merge what you have with the first subroutine and enter the next lines:

```
7            ORG    24900
20           LD     HL,BUFFER
30           LD     DE,LENGTH
220          LD     HL,BUFFER
230          LD     DE,LENGTH
450 LENGTH EQU      35000       ; LENGTH=35000
460 BUFFER NOP                  ; this is where the buffer is located
```

Also make sure that the assembler is located at about address `40000`. In any case it will be erased from memory when the sound input subroutine is called. Therefore, do not call it from a doomed assembler.

One more important note: when using these subroutines, make sure that the `BUFFER+LENGTH` expression is not greater than `65535`, otherwise some of the audio data will simply not be recorded and you will hear noise instead.

In line `100` of the first subroutine and in line `120` of the second, the delay between data samples is entered in the C register. If these two values are equal, the sound will play back at real speed. If the second number is less than the first one, the sound will be accelerated, and if it is more, it will be slowed down. Moreover, the lower the delay in the input subroutine, the higher the quality of the input signal, but the shorter its duration.

The reserve byte in the playback subroutine is designed to flip the sound, that is, play it from the end to the beginning. To enable this effect, replace the `NOP` operation in this byte with `ADD HL,DE`, and `INC HL` in line `160` with `DEC HL`.

There is another way to record sound. It is based on counting sound impulses. It does not have any particular advantages over the first method, except for the possibility of slightly lowering the noise level. Here is a subroutine that records sound using the second method:

```
10           DI                 ; interruption ban
20           LD     HL,25000    ; HL=save audio address
30           LD     DE,35000    ; DE=duration of sound
40           LD     BC,511      ; C - buffer, B - counter
50  LOOP1    IN     A,(254)     ; enter a value from port 254
60           EXX                ; register set change
70           LD     B,1         ; B=delay
80  PAUSE    DJNZ   PAUSE       ; delay
90           EXX                ; register set change
100          BIT    6,A         ; check the tape recorder bit
110          LD     A,0         ; A=0
120          JR     NZ,SIGN     ; if bit D6=1, then go to SIGN
130          DEC    A           ; A=255
140 SIGN     CP     C           ; A=C (previous value)?
150          JR     NZ,WRITE    ; If not, switch to WRITE
160          INC    B           ; B=B+1 (pulse counter)
170          JR     NZ,LOOP1    ; If B<>0, the loop
180 WRITE    DEC    B           ; B=B-1
190          JR     NZ,POKE     ; if B<>0, switch to POKE
200          LD     B,2         ; B=2
210          CPL                ; inverting A
```

```
220         LD    C,A         ; C=A
230         JR    LOOP1       ; Go to LOOP1
240  POKE   INC   B           ; B=B+1
250         LD    (HL),B      ; value recording
260         LD    B,1         ; B=1
270         LD    C,A         ; C=A
280         INC   HL          ; HL=HL+1
290         DEC   DE          ; DE=DE-1
300         LD    A,D         ; DE=
310         OR    E           ;    0?
320         JR    NZ,LOOP1    ; If not, the cycle
330         EI                ; interrupt resolution
340         RET               ; Returns
```

Line 20 specifies the sound saving address, line 30 specifies the duration of the input sound, and line 70 specifies the delay between signal samples.

Lines 180 to 240 provide noise reduction.

Register C stores the value of the previous sample, and register B serves as a counter for identical samples.

And here is the playback subroutine:

```
10          DI                ; interruption ban
20          LD    HL,25000    ; HL=saved audio address
30          LD    DE,35000    ; DE=duration of sound
40          NOP               ; reserve
50          XOR   A           ; A= border color (0)
60  LOOP2   LD    B,(HL)      ; B=next duration
70  LOOP3   OUT   (254),A     ; output A to port 254
80          EXX               ; register set change
90          LD    B,1         ; B=delay
100  PAUS2  DJNZ  PAUS2       ; delay
110         EXX               ; register set change
120         CALL  124         ; compensatory
130         BIT   0,(HL)      ;   delay
140         DJNZ  LOOP3       ; cycle
150         LD    B,(HL)      ; B=current duration
160         DEC   B           ; B=
170         INC   B           ;   0?
180         JR    Z,CONT      ; if so, switch to CONT
190         XOR   16          ; inverting the D4 bit
200  CONT   INC   HL          ; HL=HL+1
210         DEC   DE          ; DE=DE-1
220         LD    B,A         ; preservation A
230         LD    A,D         ; DE=
240         OR    E           ;    0?
250         LD    A,B         ; Recovery A
260         JR    NZ,LOOP2    ; If DE<>0, the loop
270         EI                ; interrupt resolution
280         RET               ; Returns
```

Line 20 specifies the address of the stored sound, line 30 specifies its duration, and line 90 - delay between samples (aka playback speed).

The reserve byte, as before, is for reverse playback. To enable this effect, replace the NOP operation in this byte with ADD HL,DE, and INC HL in line 200 with DEC HL.

Lines 120 and 130 are only needed for temporary synchronization with the sound recording subroutine (RET command is at address 124).

Everything said about the delay between samples and memory allocation for subroutines using the first method of recording is also true for subroutines using the second method.

The peculiarity of subroutines of the second type is that the duration of the recorded sound depends not only on the amount of memory allocated, but also on the sound itself. The fewer the level differences in the recorded sound (in other words, the lower its average frequency), the longer the sound can be recorded. For example, if you record a single "silence", then

memory is enough for a few minutes!

You can arbitrarily change the address and length of the stored sound, but you must be extremely careful not to mess up vital areas of RAM. Such as the stack or BASIC system variables.

The sound recorded by these subroutines can be saved on tape or disk using normal recording commands, and then loaded as needed.

Since even a not very long sound fragment occupies a lot of space in memory, there is a natural desire to somehow reduce it. I hasten to please you - it is possible! Audio data can be compressed very well using any algorithm. To do this, you can even use screen compressors. It would be nice to modify them a bit for such use, but if you cannot do it, I will give some tips. First, without modifying the compressor, you can compress only those files whose size is less than or equal to 6912 bytes (if any compressor refuses to accept files with a length other than 6912 bytes, then augment them to this size with zeros). Second, usually compressors when unpacking a picture place it directly in the screen area. Therefore, you will have to move the data from there to their "native" address. This can be done with the following subroutine:

```
10          LD    HL,16384    ;
20          LD    DE,16385    ; cleaning
30          LD    BC,6911     ;
40          LD    (HL),L      ; screen
50          LDIR              ;
60          CALL  DECOMP      ; decompressor call
70          LD    HL,16384    ; HL=where to move from
80          LD    DE,ADDR     ; DE=where to move
90          LD    BC,LENGTH   ; BC=how much to move
100         LDIR              ; data migration
110         CALL  3435        ; CLS subroutine call
120         RET               ; Returns
```

The data will be on the screen for such a short time that you probably won't even notice it (with a fairly rapid decompression procedure).

Finally, here is the text of a BASIC program that loads and plays back the sound with optimal parameters:

```
10 CLEAR VAL "24899": PRINT "Wait please..."
20 FOR A=VAL "24900" TO VAL "24972": READ D: POKE A,D: NEXT A
30 CLS: PRINT "Press any key, then speak...": PAUSE NOT PI
40 CLS: RANDOMIZE USR VAL "24900"
50 CLS: PRINT "Press any key to replay...": PAUSE NOT PI
60 CLS: RANDOMIZE USR VAL "24934": GOTO VAL "30"
70 DATA 243, 33, 141, 97, 17, 114, 158, 6, 8, 203, 38, 219, 254, 203, 119, 40, 2,
   203, 198, 14, 1, 13, 32, 253, 16, 239, 35, 27, 122, 179, 32, 231, 251, 201
80 DATA 243, 33, 141, 97, 17, 114, 158, 0, 62, 7, 6, 8, 230, 239, 203, 6, 48, 2,
   246, 16, 211, 254, 14, 1, 13, 32, 253, 16, 239, 35, 27, 79, 122, 179, 121,
32, 229, 251, 201
```

If you want to enable "reverse" playback, insert the following line into this program:

```
45 POKE VAL "24941",VAL "25": POKE VAL "24963",VAL "43"
```

To disable this effect, delete this line and restart the program.

The unusual way of writing numbers in this program leads to memory saving. Thus, the expression "24900" occupies 11 bytes, and "VAL "24900"" - only 8 ("NOT PI" is zero).

4.7. Reverberation.

In human terms, reverberation is an echo.

The program that realizes this interesting effect is essentially a reunion of both fragments of the first type from the previous chapter. That is, this program simultaneously plays back the sound and records a new one in its place.

Here it is:

```
10          DI                ; interrupt denial
20 LOOP1    LD    HL,25000    ; HL=buffer address
```

```
30            LD     DE,2560      ; DE = echo delay
40  LOOP2     LD     B,8          ; B= bit counter
50  LOOP3     XOR    A            ; A= border color (0)
60            RRC    (HL)         ; scrolling data through the CY flag
70            JR     NC,NOOUT     ; If CY=0, then switch to NOOUT
80            OR     16           ; setting of bit D4 of register A
90  NOOUT     OUT    (254),A      ; output A to port 254
100           RES    7,(HL)       ; reset D7 bit in memory
110           XOR    A            ; A=0 (polling the entire keyboard)
120           IN     A,(254)      ; enter a value from port 254
130           BIT    6,A          ; check the tape recorder bit
140           JR     Z,NOINP      ; if 0, then switch to NOINP
150           SET    7,(HL)       ; set bit D7 in memory
160  NOINP    CPL                 ; key
170           AND    31           ;   pressed?
180           JR     NZ,EXIT      ; if yes, go to EXIT
190           LD     C,2          ; C=delay
200  PAUSE    DEC    C            ; C=C-1
210           JR     NZ,PAUSE     ; If C<>0, the loop
220           DJNZ   LOOP3        ; continue the byte cycle
230           INC    HL           ; HL=HL+1
240           DEC    DE           ; DE=DE-1
250           LD     A,D          ; DE=
260           OR     E            ;    0?
270           JR     NZ,LOOP2     ; If not, switch to LOOP2
280           JR     LOOP1        ; Go to LOOP1
290  EXIT     EI                  ; interrupt resolution
300           RET                 ; Returns
```

This subroutine will run as long as you don't press any key. And since it is very sensitive to these kinds of actions, it is better to call it with a small BASIC program:

```
10 FOR I=1 TO 10: NEXT I
20 RANDOMIZE USR A
```

- where A is the address of the reverb subroutine.

If you want the buffer to be right behind the subroutine regardless of its location, enter the following lines:

```
20            LD     HL,BUFFER
310 BUFFER NOP                    ; this is where the buffer is located
```

Line 190 contains the value of the delay between the signal samples. By changing it, you can adjust the ratio "sound quality - memory capacity".

By changing the value in line 30 you can change the echo delay.

One last thing: if you input sound from the output of any device, the echo will sound once. If using a microphone, you can set the number of repetitions by adjusting the volume of the played sound and the recording level of the recorder. If at least one of these parameters is lower than the required level, the echo will sound once. The higher these parameters are, the more repetitions you will hear (their volume will, as in life, decrease from time to time). However, you should not get carried away. If you overdo it, the program will self-excite and everything it plays will later sound in the background of a terribly nasty noise. If this happens, turn down the volume or the microphone for a few seconds and everything will go back to normal.

4.8. Speech synthesis.

It is possible to synthesize speech simply by recording and playing it back like any other sound using the subroutines in Chapter 4.6, but this is not usually how it works. First, all the phonemes of a particular language are entered into the computer, then the user is prompted for a phrase, and finally the computer sequentially reproduces the phonemes included in it. This method saves a lot of memory and allows you to reproduce absolutely any phrase. Here is a program which allows the Spectrum to talk:

```
10 CLEAR 25343: LET L=3
```

```
20 PRINT "Wait please..."
30 FOR A=25344 TO 25433: READ D: POKE A,D: NEXT A
40 FOR A=0 TO 25: POKE 25437,A*L+100
50 CLS: PRINT "Press any key, then say '";CHR$ (A+65);'": PAUSE 0
60 RANDOMIZE USR 25344: NEXT A
70 CLS: INPUT "Enter phrase:"; LINE A$: PRINT A$
80 FOR A=1 TO LEN A$
90 IF A$(A)=" " THEN POKE 25433+A,32:NEXT A
100 IF A$(A)>="a" THEN LET A$(A)=CHR$(CODE A$(A)-32)
110 POKE 25433+A,(CODE A$(A)-65)*L+100: NEXT A: POKE 25433+A, 0
120 RANDOMIZE USR 25406: GOTO 70
130 DATA 243, 33, 0, 0, 0, 17, 0, L, 6, 8, 203, 38, 219, 254, 203, 119, 40, 2, 203,
   198, 16, 244, 35, 27, 122, 179, 32, 236, 251, 201
140 DATA 243, 33, 0, 0, 0, 17, 0, L, 62, 7, 6, 8, 230, 239, 203, 6, 48, 2, 246, 16,
   211, 254, 16, 244, 35, 27, 79, 122, 179, 121, 32, 234, 251, 201
150 DATA 33, 90, 99, 99, 126, 183, 200, 254, 32, 32, 7, 6, 5, 118, 16, 253, 24, 8,
50,
   32, 99, 229, 205, 29, 99, 225, 35, 24, 231
```

Let me explain some of the lines:

30 - are read into the program memory in the codes
40 ... 60 - phonemes are recorded from the microphone
70 - phrase input
80 ... 110 - the phrase is memorized in a special format
120 - the playback subroutine is called
130 ... 150 - subprograms in codes

After starting, the program will ask you to wait for a while while it reads the code part, then you will have to speak 26 phonemes into the microphone in sequence. You can then enter any phrases you like and enjoy them. Uppercase and lowercase Latin letters and spaces are acceptable when entering these phrases. How all other characters will sound is up to you.

unknown. Under no circumstances should you enter the " @ " sign, otherwise the program will simply
"hangs."

This program is written as an example, so it does not contain additional servo features (such as changing a single phoneme), but you are free to add them.

To change the duration of a single phoneme, change the value of the L variable in line 10 (in this case it can be from 1 to 6).

If you were not able to pronounce all the phonemes with satisfactory quality the first time, you may try again by stopping the program with the BREAK key and issuing the GO TO 40 command. When the phonemes have been recorded you may save them on tape by typing SAVE "name" CODE 25600,6656*L (the number 6656 is obtained by multiplying the number of phonemes (26) by 256). It is best if the above program with the following line is written before this block:

```
35 LOAD "name" CODE 25600: GOTO 70
```

You can record this program with the command SAVE "SPEAKER" LINE 10.

Unfortunately, 26 phonemes is not enough even for English, let alone Russian. Of course, you can increase their number, but it will also increase the amount of memory they take up. There's nothing you can do about it.

If you want your ZX-Spectrum to speak Russian, you must plug in a Russian font (see [1] for details) and increase the number of phonemes. And also change all the messages displayed on the screen.

The number of Russian phonemes can be significantly reduced by not using soft consonants (their softness is almost inaudible in the computer version) and the vowels "E", "Y", "Yu" and "Ya" (instead of them, enter "Y" and, respectively, "E", "O", "U" and "A"). Also replace the hard sign with the letter "Y".

And in conclusion, a word of advice. It is much more pleasant to hear normal human speech than words pronounced according to the rules of grammar. So enter the words as you hear them: Sinclair is sinkler, computer is camper.

4.9. Sound on interruptions.

All of the subroutines in the previous chapters have one common disadvantage: the execution of the main program is suspended while they are sounding. This can be corrected, albeit with difficulty, by using the second interrupt mode. Those who know what this is can safely skip the next few paragraphs.

As mentioned before, fifty times per second the processor receives a signal to call an interrupt. In this case it calls some subprogram and then continues to process the main program.

There are three different interrupt modes 0, 1 and 2, which are selected by the commands I M 0, I M 1 and I M 2. The standard mode is number 1. This has already been explained in Chapter 3, but just in case, let me remind you that this mode uses the ROM subroutine at address 56 (#38), which monitors the keypad and the time counter, as an interrupt handler. We are not interested in mode 0, because in ZX-Spectrum it is the same as mode 1. But mode 2 is the most interesting!

In the second interrupt mode the following happens fifty times per second: The microprocessor reads a byte from the data bus called an interrupt vector. This is sent to the low byte of the address bus, and the high byte is written to the contents of register I. At this address the processor reads two bytes from memory, which are interpreted as the address of the interrupt routine.

ZX-Spectrum is designed so that the interrupt vector is usually 255 (#FF), but some external devices such as the A M X mouse can generate other vectors. In addition, in some low-quality spectrums the interrupt vector can change completely randomly.

Based on all of the above, we can suggest the following sequence of steps to set up your own interrupt routines:

1. disable interruptions
2. write the address of the interrupt handler into memory
3. set the high byte of the pointer address to the handler in the I register
4. set the second interrupt mode
5. allow interruptions

And to return to the standard mode of interrupt handling you need to perform the following actions:

1. disable interruptions
2. write number 63 in register I
3. set the first interrupt mode
4. allow interruptions

Since the interrupt vector can change, instead of writing two bytes to a certain address, an entire table of 257 bytes is built so that the same address is read at any vector value. It is clear that all bytes of the table must be the same for this to work.

There are some rules to follow when composing interrupt handling procedures. First, the interrupt handler must be executed in a reasonably short period of time. Secondly, all registers used in the interrupt handler must assume the values they held before the interrupt was called before returning. For this reason it is not recommended to call ROM routines, at least not until you know exactly which registers are in use and which system variables can be changed in the process. Calling ROM routines is also undesirable because some of them permit interrupts, which is completely inappropriate in order to avoid calling a handler from itself. Any handler must work with forbidden interrupts. However, it is not necessary to use the DI command at the beginning of the procedure because this is done automatically and you only need to take care of allowing interrupts before exiting.

If you don't want to lose the features provided by the standard interrupt handler, you can use the RST 56 command in your subroutine. And if you use

interrupts in BASIC programs is necessary, otherwise the keyboard will be blocked.

Now let's see how we can benefit from interruptions.

If you set some sequence of sounds and play a short part of it in each interrupt, you get a pretty good effect of sound parallel to the program. And each part must be very short indeed, otherwise the interrupts will be useless.

Let's assume that the sequence of sounds will be specified by a data block in the following format: there should be two bytes in the data block for each note. The first byte is the frequency (1 ... 253), and the second byte is the duration (0 ... 255). In addition, such control codes can occur:

0 - Tone/noise switching
254 – cycle start
255 – end of block

The program is initially set to output a pure tone, but if you need to get noise, you can switch it to noise playback by inserting a byte equal to 0 in the data. To switch to tone again, byte 0 must be encountered again.

When the program encounters byte 255, the sound output will either stop or the whole sequence will repeat again, depending on the number of repetitions set.

If a code 254 is encountered in the data block, the next repetition of the effect will not start from the beginning, but from the place where this code was encountered.

Now the program itself. It is a complete package and can be translated by the assembler without the slightest change:

```
10          ORG   60000
20          JP    SINIT        ; interrupt connection
30          JP    SSTOP        ; disabling interrupts
40          JP    NEWFX        ; initializing effect
50 MUTE     LD    (COUNT),A    ; "stub"
60          RET
70 SINIT    XOR   A
80          LD    (COUNT),A
90          LD    A,24         ; JR command code
100         LD    (65535),A
110         LD    A,195        ; JP command code
120         LD    (65524),A
130         LD    HL,INTR      ; HL=handler address
140         LD    (65525),HL
150         LD    HL,65024
160         LD    DE,65025
170         LD    BC,256
180         LD    (HL),255     ; interrupt address - 65535
190         LD    A,H
200         LDIR               ; filling the table
210         DI
220         LD    I,A
230         IM    2
240         EI
250         RET
260 SSTOP   DI                 ; disabling interrupts
270         LD    A,63
280         LD    I,A
290         IM    1
300         EI
310         RET
320 NEWFX   DI                 ; initialize effect
330         LD    (COUNT),A
340         XOR   A
350         LD    (FLAG),A
360         LD    (ADDR),HL
370         LD    (CURADD),HL
380         EI
```

```
390         RET
400 ADDR    DEFW  0            ; starting address of data block
410 CURADD  DEFW  0            ; current address in the data
                                 block
420 COUNT   DEFB  0            ; repetition rate
430 FLAG    DEFB  0            ; tone/noise flag
440 INTR    PUSH  AF           ; interrupt handler
450         PUSH  BC           ; register saving
460         PUSH  DE
470         PUSH  HL
480 TEST    LD    A,(COUNT)    ; A=repetition counter
490         OR    A            ; have something to play?
500         JR    Z,EXIT
510         LD    HL,(CURADD); HL=current address
520 NEXT    LD    A,(HL)
530         INC   HL
540         CP    254          ; A=254? (cycle start)
550         JR    NZ,CONT1
560         LD    (ADDR),HL    ; change of starting address
570         JR    NEXT
580 CONT1   CP    255          ; A=255? (end)
590         JR    NZ,CONT2
600         LD    HL,(ADDR)    ; remediation of initial
610         LD    (CURADD),HL ; data block addresses
620         LD    HL,COUNT
630         DEC   (HL)         ; Repetition counter reduction
640         JR    TEST
650 CONT2   OR    A            ; A=0? (switch)
660         JR    NZ,CONT3
670         LD    A,(FLAG)
680         CPL                ; inverting A
690         LD    (FLAG),A
700         JR    NEXT
710 CONT3   LD    B,A          ; B=frequency
720         LD    C,(HL)       ; C=duration
730         INC   HL
740         LD    (CURADD),HL ; saving the current address
750         LD    A,(FLAG)     ; A=flag
760         OR    A            ; A=0?
770         LD    A,7          ; A=color of the border
780         JR    NZ,NOISE
790 TONE    XOR   16           ; tone reproduction
800         OUT   (254),A
810         PUSH  BC
820 PAUSE   DJNZ  PAUSE
830         POP   BC
840         DEC   C
850         JR    NZ,TONE
860         JR    EXIT
870 NOISE   LD    HL,1000      ; noise reproduction
880         LD    D,A
890 NOIS2   LD    A,(HL)
900         AND   248
910         OR    D
920         OUT   (254),A
930         PUSH  BC
940 PAUS2   DJNZ  PAUS2
950         POP   BC
960         INC   HL
970         DEC   C
980         JR    NZ,NOIS2
990 EXIT    POP   HL           ; register reconstruction
1000        POP   DE
1010        POP   BC
1020        POP   AF
1030        RST   56           ; Calling the standard handler
```

```
1040         RET              ; return
```

The order of use of this package should be as follows. At the beginning you need to call the subroutine SINIT, which will enable the 2nd interrupt mode. At the moment when you want to get the sound you have to put the address of the data block into the HL register and the number of repetitions into the A register and call the NE W FX subroutine. Then if you need to mute or lengthen the sound for some reason, set the new number of repetitions in register A and call the M UTE subroutine. Also, this subroutine

can be used to turn on the last sounding effect. At the end of the run (or if the program calls the drive) you should call the SSTOP subroutine, which will restore the standard interrupt mode.

The JP command series at the beginning of the package is made for convenience. It allows all subprograms of this package to be called from adjacent addresses:

SINIT - 60000 (#EA60)
SSTOP - 60003 (#EA63)
NE W FX - 60006
(#EA66) M UTE -
60009 (#EA69)

Unfortunately, it is impossible to get a sufficiently clear tone with interrupts.
Therefore, and because the data format is not musical at all, this program can hardly be used for music creation. But for sound effects it is just right.

Here is an example of how to use this program:

```
10           ORG   50000
20           CALL  60000
30           LD    HL,SNDFX
40           LD    A,3
50           CALL  60006
60           RET
70 SNDFX     DEFB  200,5,250,4,200,5,100,10,75,13,50,20,255
```

Line 70 can be replaced by the following:

```
70           SNDFXDEFB 50,20,75,13,100,10,200,5,250,4,50,20,255
```

One last thing. If your program does not have a lot of moving objects on the screen and the duration of the sound effect is short enough (about half a second or less), you can use it without any interruptions – the delay will not be noticeable.

5. Operator PL AY.

This chapter begins the part of the book devoted to the AY-3-8910 (or AY-3-8912) music coprocessor built into the ZX-Spectrum 128, as well as all its modifications.

When you turn on the ZX-Spectram 128, a menu appears on the screen, offering several options to choose from. All of them are well described in [2], but in our case select 128 BASIC. In this mode, the PLAY statement, which serves the music coprocessor, becomes available to you. In general, the format of the PLAY statement looks like this:

PLAY A$[,B$,C$,D$,E$,F$,G$,H$]

- where A$ ... H$ are character strings specifying the music program. The coprocessor itself uses only A$, B$ and C$ to control its three channels (A, B and C). If three channels are a lot for you, you can use only two, or even one, by specifying the appropriate number of strings (an empty string also indicates that the channel is not in use).

The remaining lines (D$ ... H$) are for controlling musical instruments, which can be connected to ZX-Spectram 128 via M IDI interface.

All set strings are played simultaneously, which allows you to create complex and beautiful melodies.

The thong uses special commands. Let's dwell on them in more detail.

The notes DO, RE, MI, FA, SO, LA and SI of the current octave are indicated by lowercase Latin letters, respectively, c, d, e, f, g, a and b (according to the international standard). The notes of the next octave are denoted by capital Latin letters C, D, E, F, G, A and B. For example, the operator

PLAY "cdefgabC"

will play a scale in B major.

The semitones are indicated by # (sharp) and $ (flat) in front of the note. For example, B-sharp will be written as #c, while C-flat will be written as $b. To play a B-minor scale, enter

PLAY "cdefga$bC".

It is possible to use more than one consecutive B or D. For example, the note d can be written as ##c.

The duration is determined by a number from 1 to 9 in front of the note (Table 3). The duration you set applies to all subsequent notes and pauses (see the & command). The default setting is 5 (one-fourth).

Число	Нота
	Одиночные ноты
1	Одна шестнадцатая
2	Одна шестнадцатая с точкой
3	Одна восьмая
4	Одна восьмая с точкой
5	Одна четвертая
6	Одна четвертая с точкой
7	Одна вторая
8	Одна вторая с точкой
9	Целая
	Триоли
10	Одна шестнадцатая
11	Одна восьмая
12	Одна четвертая

Табл. 3. Длительности нот в операторе PLAY.

In addition to regular notes, you can also specify triplets. Numbers from 10 to 12 are used for this purpose.

The triplet notes immediately follow the number. For example:

PLAY "3fed&11fed&fed"

Triplets do not change the set duration of the notes.

In addition to the standard note durations, you can also use arbitrary ones. To do this, put several parameters in front of the note through the underscore character ("_") to specify the desired duration. For example, you can write a 3/8 (1/8 +1/4) DO note as "3_5s".

The duration of all subsequent notes and pauses determines the last parameter in the bundle.

You get rather strange effects when you combine the durations of regular notes and trios (for example: "3_12cde"). Experiment with this. That's pretty much it. Now about the commands:

O Octave change. The octave number is given by the number from 0 to 8 following the command (the default setting is octave 5). The correspondence of the O command parameter to musical octaves is shown in Table 4.

Число	Октава
0	?
1	Субконтроктава
2	Контроктава
3	Большая октава
4	Малая октава
5	Первая октава
6	Вторая октава
7	Третья октава
8	Четвертая октава

Табл. 4. Октавы в операторе PLAY.

Note that notes below b of octave 1 will not play correctly without the M IDI interface. It is also important that although the maximum octave number is eight, there is nothing to stop you from using the next ninth octave with capital letters.

N Parameter separator. Used to separate two numeric parameters from each other. For example, when you want to specify the duration of a note right after the octave setting: "O4N5d". In essence, this command is redundant because you can use a space instead: "04 5d".

& Pause. The duration of the pause is set in the same way as the duration of the note (see above). For example: "6c& de3& de.

V Volume setting. The volume is set by a number from 0 (minimum - off) to 15 (maximum). For example: "V10". The default volume setting is 15.

W Effects programming. Notes can be played not only at a fixed volume, but also with all kinds of effects: fades, bursts, etc. The nature of the effect is set by a number from 0 to 7, according to Table 5. The default setting is 0.

X Time parameter of the sound effect. For effects 0 ... 3 the parameter sets the action duration, for 4 and 5 - period, for 6 and 7 - half-period (see table 5). The parameter value is selected from the range 0 .. 65535. The default setting is 65535.

Параметр	Эффект	Диаграмма
0	Спад, затем тихо	
1	Подъем, затем тихо	
2	Спад, затем громко	
3	Подъем, затем громко	
4	Повторяющийся спад	
5	Повторяющийся подъем	
6	Повторяющийся подъем-спад	
7	Повторяющийся спад-подъем	

Табл. 5. Программирование эффектов.

U Turns on the sound effect. After this command, all notes will play with the effect set with the W and X commands. The effects are turned off when the string is complete, or when the volume is changed (V command).

The following program will demonstrate the effect of all effects:

```
10 FOR A=0 TO 7
20 PLAY "UX1000W "+STR$ A+"cdef&"
30 NEXT A
```

Unfortunately, the musical coprocessor has only one envelope generator, so you will not be able to tune several channels to different effects simultaneously. However, you can use the same effects in several channels at once, and you only need to set the envelope parameters in one of them, and in the others it is enough to specify the U command (see above).

T Runtime. Set by a number from 60 to 240. By default, the tempo is set according to the command T120. You can only set the tempo for the whole melody, so it is defined only in the string of channel A (the T command is ignored in other channels).

(.) Repetition (reprise). A bracketed musical phrase will be repeated twice. It is allowed to use brackets in brackets (up to 4 attachments). A closing bracket without a matching opening bracket will cause the musical phrase to be repeated indefinitely from the beginning of the string. This is used, for example, in bass and percussion parts.

H Stops the PLAY operator. If this command is encountered in the string of any channel, the PLAY operator will end its work. The command is used, for example, to exit a "looped" bass part when the main tune ends.

!.! Comment. Used to insert explanations. For example:

```
PLAY "!SCALE:!cdefgabC".
```

If there are no commands in the string after the comment, the closing sign "!" is not necessary.

M Sets the mode of the channels. Each channel can play not only pure tone, but also so-called white noise. The mode distribution is set by a number from 1 to 63, determined by the sum of the codes corresponding to the mode of each channel (Table 6). Like the envelope generator, the noise generator in the musical coprocessor has only one, so you cannot tune several channels simultaneously to noise at different frequencies. By default, all three channels are tuned to tone (M 7 command). In the following example, channel A is set to tone and channel B is set to noise:

```
PLAY "M17cegbdfaCH", "O3cC)".
```

If you specify the MO command, all channels will be silent and you will hear a crackling sound.

Канал	A	B	C
Тон	1	2	4
Шум	8	16	32

Табл. 6. Режимы работы каналов.

Y Set the M channel of the IDI interface. A number from 1 to 16 following the command specifies the interface channel number to which the music data output should be routed.

Z Transmission of M IDI control codes to the M IDI interface. The M IDI code (a number from 0 to 255) must be placed immediately after the command. It is acceptable to transmit two bytes at once. In this case their values must be separated from each other with a space or N command.

Note that all PLAY operator commands must be typed in capital letters!

Finally, here is a brief summary of the PLAY statement commands (Table 7) and the error messages that may occur during its execution (Table 8).

Команда	Действие
a-g,A-G	Ноты текущей и следующей октав
H	Останов оператора PLAY
M	Устанавливает режимы работы каналов (1-63)
N	Разделитель параметров
O	Устанавливает текущую октаву (0-8)
T	Устанавливает темп (60-240)
U	Включает звуковые эффекты
V	Устанавливает громкость (0-15)
W	Задает вид звукового эффекта (0-7)
X	Задает длительность звукового эффекта (0-65535)
Y	Устанавливает канал MIDI-интерфейса (1-16)
Z	Передает данные MIDI-интерфейсу
1-12	Длительность нот или пауз
!...!	Комментарий
(...)	Репризы
$	Бемоль
#	Диез
&	Пауза
_	Соединитель длительностей

Табл. 7. Команды оператора PLAY.

Код	Сообщение	Значение
D	Too many brackets	Превышено число вложений скобок
K	Invalid note name	Встречена неопознанная команда
L	Number too big	Указано недопустимое значение параметра
M	Note out of range	Нота вышла за диапазон муз. сопроцессора
N	Out of range	Числовой параметр слишком мал или велик
O	Too many tied notes	Слишком много длительностей связано знаком "_"

Табл. 8. Ошибки оператора PLAY.

5.1. Creating Effects on PLAYe.

With the PLAY operator you can create very interesting sound effects. This is much easier to do than in the case of VEER. The number of possible effects is limited only by the programmer's imagination. Most often they use volume effects, multivoicing and a noise generator.

Since the number of effects in this case is incredibly large, I will not describe each of them (the principles are the same as in chapter 2.1), but will give a few examples with the necessary comments.

Example one:

```
10 PLAY "X16384W0O6U9B"
```

- plays a smoothly fading note.

Example two:

```
10 PLAY "O6X1000W3U1BX16384W0 9B"
```

- A modification of the previous effect. The volume of the note first increases from minimum to maximum (soft attack), and then fades smoothly.

Example three:

```
10 PLAY "T240W0X3000UO8(B&c&)"
```

- This effect is almost completely similar to the previous ones, but sounds completely different.

Example four:

```
10 PLAY "O4X16384W3U9B"
```

- Here we use an effect that in music is called a soft attack of sound.

These four effects use pure tone, but you can also use noise by inserting the M8 command, or a mixture of tone and noise (M 9 command). The following example is like this:

```
10 LET A$="T240"
20 FOR A=1 TO 8
30 LET A$=A$+"O "+STR$ A+" 1CDEFGAB
40 NEXT A: PLAY A$
```

In this effect, the music program is prepared first and then played. It could be set directly in the PLAY statement, but then it would look awful (several lines of the same text!). If you play it at the same time as creating it (in a loop), there will be undesirable pauses.

Before setting the basic data, the T240 command is written to the A$ variable, which allows you to speed up the effect. Try adding M 8 or M 9 in addition to T240.

The following example:

```
10 LET A$="T240": LET B$=""
20 FOR A=1 TO 8
30 LET A$=A$+"O "+STR$ A+" 1CDEFGAB
40 LET B$=B$+"O "+STR$ (9-A)+" 1BAGFEDC"
50 NEXT A: PLAY A$, B$
```

This effect is very similar to the previous one. But in this case two voices are used, whose frequencies are shifted in opposite directions. Another example:

```
10 PLAY "O3 1cdefgabC", "O5 1cdefgabC", "O6 1cdefgabC"
```

All three channels are used here, tuned to different frequencies, which sounds quite interesting.

The last example uses an unconventional envelope generator. It is used to form an unusual timbre:

```
10 PLAY "W4X1UcdefgabCW7X4cdefgabC"
```

The possibilities of the PLAY operator are not limited to these effects. You can experiment with it.

And in conclusion, a bit of criticism. The PLAY operator, in spite of all its advantages (which belong more to the musical coprocessor), has disadvantages. For example, it does not allow you to play notes in legato (fused) mode, and also very short notes.

5.2. Creating music on PLAYe.

The PLAY operator makes it very easy to create good three-voiced melodies.
All of its features are described in Chapter 5, so here I will limit myself to a few examples:

```
10 PLAY "T60W0X10000U3g4g1CCb5b3f4f1aag5g3eag1g#f5f3feda5g"
10 PLAY "T180W0X10000U7c6e3f9&7c6e3f&O4efefe5g&",
   "O3U3c&&&e&&f&E&F&6G3c&&&e&&f&O4EFEFE5G3c"
```

The melody can be varied by using percussion instruments. For example, the well-known song "There Was a Grasshopper in the Grass" in the style of punk rock:

```
10 PLAY "T240M35fcfcffee&ececeff&fcfffee&eceffH",
   "V14O4&f&c&e&c&e&c&f&c&f&f&c&e&c&c&f", "W0X2000Uc&)"
```

Although you can create good music and effects with the PLAY statement, the most beautiful versions are in the codes. That's why I urge you to study the next chapter.

6. Music coprocessor control.

The AY-3-8910 (AY-3-8912) music co-processor allows you to generate three-channel sound with variable volume and noise effects. This chip contains sixteen registers that control the sound. The registers are called R0 ... R15.

Select a register by writing its number to port 65533 (#FFFD) and then reading the contents of the selected register from the same port, or writing a new value of the selected register to port 49149 (#BFFD). By selecting a register number once, you can perform as many reads or writes as you want. Only when you want to access another register you have to change the contents of port 65533.

After setting the necessary parameters, the coprocessor starts generating sound, freeing the Z-80 to perform other operations.

All time intervals in the AY chip are obtained by dividing its clock frequency of 1.7734 mHz by a certain number.

6.1. Registers.

R0 – low byte of channel A frequency
R1 – high byte of channel A frequency
R2 – low byte of channel B frequency
R3 – high byte of channel B frequency
R4 – low byte of channel C frequency
R5 – high byte of channel C frequency

The desired frequency of any channel is obtained by dividing the coprocessor clock frequency by 16 and then dividing the result by the 12-bit value obtained by merging the frequency low byte register and the D0 ... D3 bits of the frequency high byte register. So, in total you can set 4095 different frequencies (from 27 Hz to 110 kHz). Of course you will not hear the highest ones. Note that the difference between two adjacent values in the low frequencies is a fraction of a hertz, and in the high frequencies it reaches several kilohertz.

The low and high bytes, as well as the frequency, can be calculated using the following formulas:

```
Hi=INT(INT(110830/Fq+.5)/256)
Lo=INT(110830/Fq+.5)-Hi*256
Fq=110830/(Lo+256*Hi)
```

- where Hi is the high byte, Lo is the low byte, and Fq is the frequency in Hz. Note that in most cases a given frequency cannot be accurately transferred to the music coprocessor. Table 9 shows the values of the tone registers for all possible notes.

НОТА / ОКТАВА	СК	К	Б	М	1	2	3	4	5	6
ДО		3389	1694	847	424	212	106	53	26	13
ДО-ДИЕЗ		3199	1599	800	400	200	100	50	25	12
РЕ		3019	1510	755	377	189	94	47	24	
РЕ-ДИЕЗ		2850	1425	712	356	178	89	45	22	
МИ		2690	1345	672	336	168	84	42	21	
ФА		2539	1269	635	317	159	79	40	20	
ФА-ДИЕЗ		2396	1198	599	300	150	75	37	19	
СОЛЬ		2262	1131	565	283	141	71	35	18	
СОЛЬ-ДИЕЗ		2135	1067	534	267	133	67	33	17	
ЛЯ	4030	2015	1008	504	252	126	63	31	16	
ЛЯ-ДИЕЗ	3803	1902	951	476	238	119	59	30	15	
СИ	3590	1795	898	449	224	112	56	28	14	

Табл. 9. Значения регистров тона.

R6 – sets the noise frequency

The required noise frequency is obtained by dividing the clock frequency by 256 and then dividing the result by the 5-digit value located in the five lowest bits (D0 .. D4) of register R6. A total of 31 different frequencies (from 223 Hz to 7 kHz) can be set.

The value of register R6 and the noise frequency can be calculated using the following formulas:

```
R=INT(6927.3437/Fq+.5)
Fq=6927.3437/R
```

- where R is the value of R6, and Fq is the frequency in Hz.

Everything said about the tone generators is also true for the noise generator.

R7 – mixer and I/O control The individual bits of this register are used for different purposes: D0 – tone of channel A

D1 – channel B tone D2 – channel C tone D3 – channel A noise D4 – channel B noise

D5 – channel C noise

D6 - port A: 0 - input/1 - output

D7 - port B: 0 - input/1 - output (AY-3-8910 only)

A zero in bits D0 ... D5 indicates that the function is enabled. Thus each channel can be configured to three different states: tone, noise and tone+noise. The I/O ports are not used in the ZX-Spectrum music coprocessor, but you can connect additional joysticks, printer, MIDI interface or something similar to them. Port B exists only in the AY-3-8910.

R8 – channel A amplitude control R9 – channel B amplitude control R10 – channel C amplitude control

Bits D0 to D3 of these registers set the volume for each channel (from 0 to 15). If bit D4 is set in one (or more) of them, the value in the lower bits is ignored, and the amplitude of this channel is controlled by the envelope generator (see next).

R11 – low byte of the envelope period

R12 – high byte of the envelope period

The required envelope period is obtained by dividing the clock frequency of the coprocessor by 256 and then dividing the result by the 16-bit value obtained by merging the registers R11 and R12. A total of 65535 different periods can be set. The values of registers R11 and R12, as well as the frequency and period of the envelope can be calculated by the following formulas:

```
Hi=INT(INT(6927.3437/Fq+.5)/256)
Lo=INT(6927.3437/Fq+.5)-Hi*256
Fq=6927.3437/(Lo+256*Hi)=1/Pd
Pd=(Lo+256*Hi)/6927.3437=1/Fq
```

- where Lo and Hi are the values of registers R11 and R12, respectively, Fq is the envelope frequency in Hz, and Pd is period of the envelope in seconds.

R13 – envelope shape control

The required shape of the envelope is set by bits D0 ... D3 of register R13 as follows: D0 – attenuation

D1 – alternation

D2 – increase D3 – continuation

Table 10 shows all possible combinations of these bits.

D3 D2 D1 D0	Значение R13	Эффект	Диаграмма
0 0 X X 1 0 0 1	0…3 9	Спад, затем тихо	
0 1 X X 1 1 1 1	4…7 15	Подъем, затем тихо	
1 0 0 0	8	Повторяющийся спад	
1 0 1 0	10	Повторяющийся спад-подъем	
1 0 1 1	11	Спад, затем громко	
1 1 0 0	12	Повторяющийся подъем	
1 1 0 1	13	Подъем, затем громко	
1 1 1 0	14	Повторяющийся подъем-спад	

Табл. 10. Значения регистра R13.

R14 - I/O port A R15 - I/O port B

As mentioned above, the I/O ports are not important and do not affect the generated sound.

6.2. Programming.

All sound effects and music are programmed by permanently changing register values with the necessary delays. This can be done, for example, by the following subroutine:

```
10          LD    HL,60000    ; Hb=data address
20 LOOP     LD    A,(HL)      ; A=byte of data
30          INC   HL          ; HL=HL+1
40          CP    255         ; A=255?
50          RET   Z           ; If so, a refund
60          CP    16          ; A=16?
70          JR    NZ,REG      ; If not, switch to REG
80          LD    B,(HL)      ; B = pause time
90 PAUSE    HALT              ; abort expectation
100         DJNZ  PAUSE       ; cycle
110         JR    CONT        ; go to CONT
120 REG     LD    BC,65533    ; BC=register port address
130         OUT   (C),A       ; write the register number
140         LD    B,191       ; BC=data port address
150         LD    A,(HL)      ; A=register value
160         OUT   (C),A       ; write data to the register
170 CONT    INC   HL          ; HL=HL+1
180         JR    LOOP        ; switch to the beginning
```

This program is rather primitive and not very suitable for playing music, but it is just fine for creating simple effects. Before running it, don't forget to prepare a data block at address 60000, consisting of data pairs ending with the number 255. The first value in each pair must be the register number and the second value must be the number you want to write to that register. In addition, if the first value is 16, the second value is interpreted as a delay (in fiftieths of a second).

For music playback, much more complex subroutines are used, usually operating in the second interrupt mode. The data for these subroutines is usually stored in a much more convenient form, separately for each of the three channels. It is not possible to give a complete example of at least the simplest subprogram in this book because of its complexity, but if you know how to work with assembler, you should not need to make up such a subprogram, and I can give you a hint on how to do it.

First, let's deal with the data format. I propose the following system: the melody will be set by three (by number of voices) main data blocks. Since almost any tune consists of the same fragments, repeated in different order, it would be logical to set not the tune itself, but the addresses of these fragments in the main blocks. These fragments are usually called patterns (pattern).

So, the main blocks can contain the following two-byte values: 65535 (#FFFFFF) - end of melody

0 (#0000) - cycle start

addr (#XXXX) - address of the next pattern

The "start of cycle" code (0) marks the point at which the melody will begin to play when it is repeated.

Now let's look at patterns. Without going into theory, here's the format I developed:

128 (#80) - end of pattern
129,n (#81,#XX) - set the duration n 130,n (#82,#XX) - noise with frequency n(0 ... 31)
131,n1,n2 (#83,#XX,#XX) - noise with frequency n (0 ... 31) + note (0 ... 100) 132,n1,n2 (#84,#XX,#XX) - direct tone frequency reference (0 .. . 4095) 133,addr (#85,#XXXX) - set tone frequency block 134,addr (#86,#XXXX) - set the block for changing the noise frequency 135,addr (#87,#XXXX) - volume block setting 136,n1,n2,n3 (#88,#XX,#XXXXX) - envelope generator control
0 ... 100 (#00... #64) -notes from the CY subcontact

Let me explain some of the codes:

129 - As you may have noticed, when specifying a note its duration is not specified. The point is that it uses the duration set in advance with this command. The duration is measured in fiftieths of a second.

131 - With this code you can play tone and noise at the same time. 133,134,135 - These codes specify additional data blocks indicating how to change The tone frequency, noise frequency and volume over the duration of the note. If there is a 0 after codes 133 or 134 instead of the block address, the frequency change is disabled. A number between 0 and 15 after code 135 indicates that the volume level corresponding to that number must be kept constant.

136 - This code controls the envelope generator. It must be followed by a one-byte number from 0 to 7, indicating the shape of the envelope according to Table 5 and a two-byte number specifying the period of change of the envelope.

0 ... 100 - These codes specify the notes. Code 0 is for the note of the CJ of the subcontact, 1 is for the CJ#, 2 is for the SI, etc.

The block describing the tone frequency change uses the following values: 128 (#80) - end of block
-127 ... 127 (#81 ... .#7 9) - frequency offsets

Noise frequency block will be set in the following format: 128 (#80) - end of block
-31 ... 31 (#E1 ... .#1F) - frequency offsets

And let the volume change block be set as follows: 128 (#80) -end of block
0 ... 15 (#00 ... .#0F) - volume values

It would be quite logical to make this procedure work in the second interrupt mode. Based on this, as well as on the proposed data format, you can understand that for each of the channels you will need an array of variables. I propose the following format:

offset	***Size***	***value***
0	*2*	*Start address of the main data block*
2	*2*	*current address in the main data block*
4	*2*	*Initial address of the tone frequency block*
6	*2*	*current address in the tone frequency block*
8	*2*	*Initial address of the noise frequency block*
10	*2*	*the current address in the noise frequency block*
12	*2*	*Initial address of the volume change block*
14	*2*	*current address in the volume change block*
16	*2*	*the current address in the current pattern*

18	1	current duration value
19	1	duration counter
offset	**Size**	**value**
20	1	number of repetitions remaining

In addition, you need three more bytes per channel to store the tempo, the tempo counter, and the sound enable flag.

This package will include the following procedures:

`SINIT` – Initialization of tables, connection of the second interrupt mode.

`SSTOP` – Turns off the coprocessor, restoring the standard interrupt mode.

`SNE W` – Starts all three channels. When this subroutine is called, the `HL`, `DE` and `BC` registers must contain the addresses of the main data blocks for channels A, B and C, respectively. Register A must contain the number of times the tune repeats from `1` to `254`, or `255 if` you want the tune to repeat infinitely.

`SNE W A` – Start channel A. The `HL` register must contain the data block address and the A register `must contain` the number of repetitions (similar to `SNE W`). The operation of channels B and C is not affected by this procedure.

`SNE W B` – Starting Channel B. Similar to `SNE W A`.

`SNE W C` – Starting Channel C. Similar to `SNE W A`.

`M UTE` – Prohibit/enable operation. The A register must contain a channel number from O to `2` or `3`, if you are accessing all channels. The B register must contain the mode code: `0` - stop, `1` – silent operation, `2` – playback.

`STATUS` – Receives the state of the channel. Register A must contain the channel number from `0` to `2`. On return from the `STATUS` procedure, register A contains the mode code for the selected channel (similar to B in `M UTE`).

`TE M PO` – Tempo setting. Register A must contain the channel number from `0` to `2`, or `3` if you are accessing all channels. Register B should contain the tempo value. So, the beginning of a packet might look like this:

```
10          ORG   60000
20          JP    SINIT
30          JP    SSTOP
40          JP    SNEW
50          JP    SNEWA
60          JP    SNEWB
70          JP    SNEWC
80          JP    MUTE
90          JP    STATUS
```

The `TE M PO` procedure might look like this:

```
100 TEMPO   DI
110         PUSH  HL
120         PUSH  AF
130         LD    HL,TEMPS    ; HL=indicator at the
                                                 rate
140         CP    A,3
150         JR    Z,TEMP3
160         ADD   A,L
170         LD    L,A
180         JR    NC,TEMP1
190         INC   H
200 TEMP1   LD    (HL),B
210         INC   HL
220         INC   HL
230         INC   HL
240         LD    (HL),B
250 TEMP2   POP   AF
260         POP   HL
270         EI
280         RET
290 TEMP3   PUSH  DE
300         LD    D,6
```

```
310 TEMP4   LD     (HL),B
320         INC    HL
330         DEC    D
```

```
340         JR    NZ,TEMP4
350         POP   DE
360         JR    TEMP2
```

Since there are references to the data, the to bring a line with their description :

```
370 CHAN_A DEFS 21                      ; an array of variables for    A
                                        the channel
380 CHAN_B  DEFS  21                    ; an array of variables for    B
                                          the channel
390 CHAN_C  DEFS  21                    ; an array of variables for    C
                                          the channel
400 MUTS    DEFS  3                     ; sound resolution flags
410 TEMPS   DEFS  3                     ; rates
420 CURTS   DEFS  3                     ; pace meters
430 AYREGS  DEFS  14                    ; coprocessor registers
440 ENVS    DEFB  0,4,11,13,8,12,14,10  ; envelope shape
450 SVOLS   DEFW  #8000,#8001,#8002     ; volume tables
460         DEFW  #8003,#8004,#8005     ; for standard values
470         DEFW  #8006,#8007,#8008
480         DEFW  #8009,#800A,#800B
490         DEFW  #800C,#800D,#800E
500         DEFW  #800F
510 NOTES   DEFW  ...                   ; This is where you need to    values from
                                        ; write down all the           angle in order
                                          Table 9 from the upper left
                                        ; top to bottom, right to left
```

Line 430 contains the data area that is used for sound output. First it forms the values of all the registers of the coprocessor with the following subroutine:

```
520         SETAYPUSH HL
            530PUSH AF
540         LD    AL,AYREGS
550         ADD   A,L
560         LD    L,A
570         JR    NC,SETAY1
580         INC   HL
590 SETAY1 LD     (HL),B
600         POP   AF
610         POP   HL
620         RET
```

It needs to pass the register number in A and its value in B. Then the contents of this area are copied to the real registers of the coprocessor by another subroutine:

```
630 AYOUT   PUSH  HL
640         PUSH  DE
650         PUSH  BC
660         LD    HL,AYREGS+13
670         LD    D,13
680 AYOUT1  LD    BC,65533
690         OUT   (C),D
700         LD    B,191
710         LD    E,(HL)
720         OUT   (C),E
730         DEC   HL
740         DEC   D
750         JP    P,AYOUT1     ; if D>=0, go to AYOUT1
760         POP   BC
770         POP   DE
780         POP   HL
790         RET
```

Any of the registers in this area can also be read:

```
800 GETAY   PUSH  HL
810         PUSH  AF
820         LD    HL,AYREGS
830         ADD   A,L
840         LD    L,A
850         JR    NC,GETAY1
860         INC   H
```

```
870 GETAY1  LD     B,(HL)
880         POP    AF
890         POP    HL
```

```
900         RET
```

This subroutine needs to pass the number of the desired register to A and it will return its value to B.

Line 440 is needed to decode the envelope code. If you add any number from table 5 to the ENVS label and read one byte from the resulting address, you will get the value to write to R13.

Line 450 will be needed to decode the standard volume levels. To do so, multiply a number from 0 to 15 by 2 (you can use the SLA command) and add to the SVOLS label. The resulting value should be used as the address of the volume change block.

Line 510 will be useful for decoding notes. If you multiply the note code (from 0 to 100) by 2 (SLA command) and add to the NOTES label, and read the resulting address as a two-byte number, you will get the values of the lowest and highest frequency registers.

Now let's go over the basic procedures:

```
910 STATUS PUSH HL
920         LD    HL,MUTS
930         ADD   A,L
940         LD    L,A
950         JR    NC,STAT1
960         INC   H
970 STAT1   LD    A,(HL)
980         POP   HL
990         RET
```

This small but useful procedure will help the programmer to know in which state a particular channel is. For example, to find which one is free.

The next procedure is M UTE:

```
1000 MUTE   DI
            1010PUSH HL
            1020PUSH AF
1030        LD    HL,MUTS
1040        CP    A,3
1050        JR    Z,MUT2
1060        ADD   A,L
1070        LD    L,A
1080        JR    NC,MUT1
1090        INC   H
1100 MUT1   LD    (HL),B
1110        POP   AF
1120        POP   HL
1130        EI
1140        RET
1150 MUT2   LD    (HL),B
1160        INC   HL
1170        LD    (HL),B
1180        INC   HL
1190        JR    MUT1
```

This procedure will be needed, for example, to temporarily stop one of the channels.

Now for the channel initialization procedures:

```
1200 SNEWC  PUSH  IX
1210        LD    IX,CHAN_C
1220        PUSH  BC
1230        LD    B,2
1240        JR    SNEW1
1250 SNEWB  PUSH  IX
1260        LD    IX,CHAN_B
1270        PUSH  BC
1280        LD    B,1
1290        JR    SNEW1
1300 SNEWA  PUSH  IX
1310        LD    IX,CHAN_A
1320        PUSH  BC
1330        LD    B,0
```

```
1340 SNEW1  DI
1350        PUSH  BC
1360        PUSH  DE
1370        PUSH  HL
1380        PUSH  IX
1390        POP   HL
1400        PUSH  HL
1410        POP   DE
1420        INC   DE
1430        LD    BC,20
1440        LD    (HL),B
1450        LDIR
1460        POP   HL
1470        PUSH  HL
1480        LD    (IX+20),A
1490        LD    (IX+0),L
1500        LD    (IX+1),H
1510        LD    (IX+19),0
1520        LD    E,(HL)
1530        INC   HL
1540        LD    D,(HL)
1550        INC   HL
1560        LD    (IX+2),L
1570        LD    (IX+3),H
1580        LD    (IX+16),E
1590        LD    (IX+17),D
1600        LD    HL,SVOLS+30
1610        LD    (IX+12),L
1620        LD    (IX+13),H
1630        LD    (IX+14),L
1640        LD    (IX+15),H
1650        LD    (IX+18),13 ; default duration =           1/4
1660        POP   HL
1670        POP   DE
1680        POP   BC
1690        LD    A,B
1700        LD    B,2
1710        CALL  MUTE
1720        LD    B,1
1730        CALL  TEMPO
1740        POP   BC
1750        POP   IX
1760        EI
1770        RET
```

And finally, the initialization of all three channels:

```
1780        SNEWPUSH HL
1790        CALL SNEWA
1800        PUSH DE
1810        POP   HL
1820        CALL SNEWB
1830        PUSH BC
1840        POP   HL
1850        CALL SNEWC
1860        POP   HL
1870        RET
```

The initialization routines prepare all the data necessary for operation in the variable arrays. They set the addresses of the tone frequency and noise blocks to "not used" (put 0 in them). They select the constant volume level (15). They also set the default note duration to 1/4 second.

Here is the subroutine for connecting the second interrupt mode:

```
1880 SINIT LD     A,24
1890        LD    (65535),A
1900        LD    A,195
1910        LD    (65524),A
```

```
1920         LD     HL,INTR      ; HL=aflpec handler
1930         LD     (65525),HL
1940         LD     HL,65024
1950         LD     DE,65025
1960         LD     BC,256
1970         LD     (HL),255
1980         LD     A,H
1990         LDIR
2000         DI
2010         LD     I,A
2020         IM     2
2030         LD     HL,MUTS      ; prohibit
2040         XOR    A
2050         LD     (HL),A       ; works
2060         INC    HL
2070         LD     (HL),A       ; all
2080         INC    HL
2090         LD     (HL),A       ; channels
2100         EI
2110         RET
```

And this is the subroutine that returns the standard mode interrupts and shutting down the coprocessor:

```
2120 SSTOP   DI
2130         LD     A,63
2140         LD     I,A
2150         IM     1
2160         EI
2170         LD     HL,AYREGS
2180         LD     DE,AYREGS+1
2190         LD     WSDH
2200         LD     (HL),B       ; B=0
2210         LDIR                ; clear registers area
2220         LD     A,7
2230         DEC    B            ; B=255
2240         CALL   SETAY        ; R7=255 (mixer off)
2250         JP     AYOUT        ; output registers to the coprocessor
```

Now let's deal with the interrupt handler. Since it must handle three channels and variable arrays are most easily addressed with register IX, we can propose the following subroutine:

```
2260         INTRPUSH AF         ; saving registers
             2270PUSH HL
             2280PUSH DE
             2290PUSH BC
             2300PUSH IX
2310         LD     IX,CHAN_A    ; prepare registers
2320         XOR    A
             2330CALL DISPAT     ; coprocessor
2340         LD     IX,CHAN_B
2350         LD     A,1          ; for all channels
             2360CALL DISPAT
2370         LD     IX,CHAN_C
2380         LD     A,2
             2390CALL DISPAT
             2400CALL AYOUT      ; output registers to coprocessor
2410         POP    IX           ; registers restore
2420         POP    BC
2430         POP    DE
2440         POP    HL
2450         POP    AF
2460         RST    56           ; calling the standard handler
2470         RET
```

This handler for each of the channels calls the procedure - dispatcher (DISPAT), entering in register A channel number, and in register IX - the address of the array of its variables.

The role of the procedure DISPAT is to process the variables TE M PS, CURTS and M UTS for the specified channel, as well as in the call of the main sound creation subroutine - GETSND.

Here is the text of the DISPAT procedure:

```
2480 DISPAT PUSH AF
2490        LD    E,A
2500        LD    D,0
            2510CALL STATUS
2520        OR    A           ; channel stopped ?
2530        JR    NZ,DISP1
2540        POP   AF
2550        LD    B,A
2560        JR    DISP3
2570 DISP1 LD     HL,CURTS
2580        ADD   HL,DE
2590        LD    A,(HL)
2600        OR    A
2610        JR    Z,DISP2
2620        DEC   (HL)        ; ramp counter decrease
2630        POP   AF
2640        RET
2650 DISP2 DEC    HL          ; update
2660        DEC   HL
2670        DEC   HL          ; counter
2680        LD    A,(HL)
2690        INC   HL          ; temp.
2700        INC   HL
2710        INC   HL
2720        LD    (HL),A
2730        POP   AF
            2740CALL GETSND   ; call the main procedure
2750        LD    B,A
            2760CALL STATUS
2770        CP    1           ; must be "jammed" ?
2780        RET   NZ
2790 DISP3 LDC    ,B          ; "muting"
2800        LD    HL,AYREGS+6
2810        LD    A,9         ; channel
2820 DISP4 SLA    A
            2830DJNZ DISP4
2840        OR    (HL)
2850        LD    (HL),A
2860        INC   HL
2870        ADD   HL,BC
2880        LD    (HL),0
2890        RET
```

So, all the service procedures are listed. There is only one left - GETSND:

```
2900 GETSND ...
```

That's what I'm offering you. But don't be frightened - I will explain everything in detail.

Most likely, the GETSND procedure will be quite large. Maybe even larger than all of these procedures combined. But there is nothing difficult about it, and its size is due to a fairly complex data format.

The task of the GETSND procedure is to form in certain cells of the AYREGS data of one of the channels for subsequent copying them into the coprocessor registers.

This procedure takes the channel number in register A as a parameter (so that it knows in which cells to place data for frequency, volume, etc.) and the address of the variable array in register IX. Note that it must save the value of register A! You may even have to have an additional variable to store it.

So, the procedure in the GETSND procedure is as follows:

1. Check if the duration counter (IX+19) is not zero.
2. If not, continue playing the current note.
3. If equal, select a new note and start playing it.

The term "continue playing the current note" includes the following:

1. Decrease the duration counter.
2. If the address of one of the additional blocks is 0, then steps 3 ... 5 for this block do not need to be performed.
3. Select the next values from the frequency and volume change blocks using the variables IX+6/IX+7, IX+10/IX+11, and IX+14/IX+15.
4. According to the selected values and the channel number, update the AYREGS area using the GETAY and SETAY procedures (note that frequency offsets can also be negative).
5. Update the variables on addresses IX+6/IX+7, IX+10/IX+11 and IX+14/IX+15 according to point 3.

"Start note playback" includes only one item:

1. Rewrite the variables on addresses IX+4/IX+5, IX+8/IX+9, IX+12/IX+13 and IX+18 into their counters - duplicates (IX+6/IX+7, IX+10/IX+11, IX+14/IX+15, and IX+19).

But "select a new note" is more difficult:

1. Select the next byte from the current pattern (address is IX+16/IX+17).
2. If it is not equal to 0 ... 100, 130, 131 and 132 - treat it as a corresponding control code and go to step 1.
3. Accordingly with the byte or its parameters and the channel number update area AYREGS and variable 1X+16YX+17.

Now about control code processing:

***Code* 128:**

1. Select the address of the next pattern from the main block (the address in the main block is contained in the IX+2/IX+3).
2. Update variable IX+2/IX+3.
3. If the pattern address is zero, copy variable IX+2/IX+3 to IX+0/IX+1 and go to step 1.
4. If the address is 65535, copy variable IX+0/IX+1 to IX+2/IX+3 and decrease the repetition counter (IX+20). If the repetition counter is zero, set the channel to "stopped" using the M UTE procedure. Go to step 1.
5. Write the selected address to IX+16/IX+17 and "select new note".

***Code* 129:**

1. Take the byte following this code and put it in IX+18. "Select New Note.

***Code* 133 (134):**

1. Take the address following this code and place it in IX+4/IX+5 (IX+8/IX+9). "Select New Note.

***Code* 135:**

1. Take the address following this code.
2. If it is less than 16, calculate the corresponding address in the SVOLS table.
3. Place the received address in IX+12/IX+13.
4. "Choose a new note."

***Code* 136:**

1. Set the volume register of the specified channel to 16.
2. Write in IX+14/IX+15 zero.
3. Take the byte following this code.
4. Calculate the R13 value from the ENVS table and update the AYREGS area.
5. Take the two bytes following the form code and put them in cells 11 and 12 of the AYREGS area.
6. "Choose a new note."

Now about how you can calculate the register numbers for a given channel. To calculate the frequency register number just multiply the channel number by 2 (ADD A,A). The resulting number will be the register number of the low byte frequency. To get the register number of the high frequency byte, increase the value by 1 (IN C A).

To calculate the volume register number, add 8 to the channel number (ADD A,8).

If you write the GETSND procedure, you will have a pretty powerful program in your hands, suitable for writing both music and effects.

And at the end of the description of this program – tips on making data blocks for it.

To raise or lower a note by an octave, its value must be correspondingly increased or decreased by 12. The note DO of the first octave corresponds to the number 39.

You can take the values of durations from Table 11.

Нота	Значение
Одна шестнадцатая	3
Одна шестнадцатая с точкой	5
Одна восьмая	6
Одна восьмая с точкой	9
Одна четвертая	13
Одна четвертая с точкой	19
Одна вторая	25
Одна вторая с точкой	38
Целая	50

Табл. 11. Длительности нот.

Now some tips on programming the coprocessor.

To form new tones, you can use an envelope generator tuned to a periodically changing volume and high frequency. Particularly good results can be achieved by tuning it to a frequency multiple of the main signal frequency.

To silence the music coprocessor it is quite common to disable all mixer functions (output byte 255 in R7), but this method is not very reliable. If the envelope generator is set to a periodically changing volume, this trick will not work: you will still hear the clicking sound. To completely silence the co-processor I can advise the following subroutine:

```
10          LD    HL,DATA     ; HL=data address
20          LD    E,10        ; E=first register number
30 LOOP     LD    BC,65533    ; BC=register port address
40          OUT   (C),E       ; output E to port BC
50          LD    A,(HL)      ; A=value of the next register
60          LD    B,191       ; BC=data port address
70          OUT   (C),A       ; output A to port BC
80          INC   HL          ; HL=HL+1
90          DEC   E           ; E=E-1
100         LD    A,E         ; E=
110         CP    6           ; 6 ?
120         JR    NZ,LOOP     ; If not, the cycle
130         RET               ; Returns
140         DATADEFB 0,0,0,255     ; data for registers AY
```

In many cases it is necessary to determine whether a coprocessor is present in a given computer. Some people do it by checking the computer type (48K/128K), but this way is not really fair, because the AY can also be on good old Spectrum. Here is a subroutine to determine the presence of the coprocessor more reliably:

```
10          LD    BC,65533    ; BC= register port address
20          XOR   A           ; A=0
30          OUT   (C),A       ; register selection 0
40          LD    B,191       ; BC=data port address
50          OUT   (C),A       ; output 0 in the selected register
60          LD    B,255       ; BC=register port address
70          IN    A,(C)       ; enter a value from the selected
                                register
80          OR    A           ; A=0?
90          RET               ; Returns
```

If the Z flag is cleared after calling this subroutine, the coprocessor is present. Otherwise it is not.

And finally, I want to tell you about a rather interesting trick. It is very often used in many programs. By reading the data from the volume registers and converting them appropriately into graphical information simultaneously with playback

music, you can make color music or peak indicators of signal level.

7. Software review.

For ZX-Spectrum created quite a few programs that allow you to create various sounds, but unfortunately, many of them do not make a positive impression.

The first group of programs I will describe can be dubbed sound effects editors. They are very few in number.

OZ Software's SPECSOUND program has a very uncomfortable user interface and the ability to edit just one effect.

The SOUND FX program by Dk'tronics is a pretty good piece of work. It allows you to create effects with both pure tone and noise effects, but has one major drawback: the effects it selects itself, randomly, it is impossible to edit them in the literal sense of the word.

There is also a package DZ W IEKI, which unfortunately I could not get.
Some claim that this is the first version of the SPECSOUND program.

The SUPER SOUND program written by the author of this book (you can find its detailed description in the next chapter) allows you to edit ten different effects. Among them are both tone and noise effects. It also allows you to input sound fragments from the Spectrum's tape input and play them back in various modes.

All of the above effect editors allow you to record the edited effects on magnetic media as subroutines in codes that you can use in your programs.

I could not find any programs that allow you to create sound effects for the music coprocessor at all. But this will be fixed with the next version of SUPER SOUND, which is expected quite soon.

The next group of programs allows you to record and play back audio fragments via the ZX-Spectrum tape input. SPEAK EASY is typical, and the leader is VOICE M ANIPULATO R, written by Julian Spencer in 1991.

There are two disadvantages to these kinds of programs: the sound quality is mediocre and their length is limited to a few tens of seconds (see chapter 3.5).

Quite interesting are the programs that allow your Spectrum to speak human language. However, their pronunciation suffers greatly and they speak with an accent. Perhaps the whisperiest of them all is BASZED. You can understand what she is saying only by reading the phrase on the screen. It is even popularly dubbed "bazoom". The FONGEN (Phoneme Generator) program is a bit more understandable. In addition, it can be made to speak in Russian.

The list does not end there. There are other such programs: L M O W A, TOKER, etc.

There is also, in my opinion, a one-of-a-kind digital frequency meter - the DIGITAL FREQUENCER (DFR). Although the idea is not bad, the design is ugly.

Several programs allow you to see spectrum diagrams of the sound signal fed to the ZX-Spectram tape input. For example, TAPER and TARE DIAGNOSTICS can decompose the sound into frequencies up to 4 kHz. The LIGHT SHO W program, written by Ziga Turk in 1984, allows you to analyze sound up to 16 kHz. It also has a color-music mode that creates color effects on the screen in time with music fed to the tape input.

The next group of programs are music editors. There are quite a few of them. The simplest of them, such as M enzer Synthetizer or Organ, make the ZX-Spectram something like a child's squeaker, with the Spectrum keys as the keyboard. More sophisticated programs, such as Spectrum M usicmaker or M usic Typewriter, allow you to save a tune on a tape, with subsequent loading and playback.

Einstein Software's A.E.Drums program specializes in percussion instruments. It contains ten different percussion instruments and as many ready-made rhythms.

One of the most powerful music editors is W HA M THE M USIC BOX,

created by M ark Soft in 1985. It allows you to write two-voice melodies with percussion instruments. You will find its detailed description in one of the following chapters.

It is suspected that there are two excellent music editors, superior in their capabilities to W HA M THE M USIC BOX, but inaccessible to the average user. You can hear the results of one of them in such programs as STAR W ARS, GOLDEN AXE, GR AND PRIX, and OPERATION W OLF. The music from the other one I was able to find only in the programs of Code M asters (T W IN TURBO V8, RALLY CROSS), which suggests some thoughts.

There are quite a few music editors for the coprocessor as well. The two most famous are W HA M THE M USIC BOX 128K and ASC Sound M aster (AS M). The first is written by the same firm M ark Soft, and the second by Andrew Sendetsky of Dnepropetrovsk (firm Andrew Strikes Code). Although, W HA M 128K is one of the best music editors for coprocessor, it is still far from AS M. You can find a description of AS M in the SPECTROFON electronic magazine #2.

Perhaps the last group of sound programs are the music demonstration programs for the coprocessor. These can be divided into two subgroups. The first is music compilations, and the second is dynamic shows.

Typical representatives of the first subgroup are, for example, M ANHATTAN or TOR 128. These programs are not very interesting, but they are large libraries of tunes.

From the second subgroup two of the most beautiful programs can be distinguished. They are SHOCK and THE LYRA II of the Polish firm ETHANOL SOFT INC. Each of them contains several parts (8 and 9 respectively) with beautiful graphics and music.

This review does not pretend to be complete, but I have tried to cover as many types of programs as possible.

7.1. SUPER SO UND sound effects editor.

The SUPER SOUND v2.2 program allows you to select and customize one of eleven different sound effects for your own needs and then offload them to disk or tape, depending on the version. All files unloaded by SUPER SOUND are fully relocatable, complete subroutines in machine code, and can be called from BASIC as well as assembler.

SUPER SOUND works on all types of Spectrum-compatible computers and does not need a music coprocessor (AY-3-8912).

The program is controlled with the cursor and the menu. The functions are selected by hovering the cursor over the corresponding item in the current window and pressing the "fire" button. To control the cursor you can use Kempston joystick, Sinclair joystick or keyboard (keys O, P, Q, A and SPA CE - left, right, up, down and fire respectively). All control devices are active at the same time.

The main menu contains twelve items. The first eleven are various effects, and the last (Info) is information about the program, the date it was created, and the author.

7.1.1. Effects.

I will describe all the effects one by one. When you select any of them, a window opens in front of you with the current parameters of that effect and several service functions. Each of the eleven windows has three general functions: M ain M enu - return to the main menu, Play - play the effect, and Save - write the adjusted effect to magnetic media. The other functions change from effect to effect.

Input Sound - input sound. This main menu item lets you input audio fragments fed to the computer's tape input and play them back at different speeds. In addition to the three standard functions in the effect window, there are several functions of their own: Input Speed:XX - input speed. The smaller is this parameter, the better is the quality of input audio, the more it occupies memory and the less is the number of times it can be accelerated. If you want maximum quality, the input speed should be 0, but then what you input can either be played at real speed or slowed down. Play

Speed:XX - playback speed. If the parameter of this item is equal to the Input Speed parameter, the sound fragment is played back at real speed, if it is less, the sound is accelerated, if it is more, the sound is slowed down. Length:XXXXX - the length of sound fragment. Reverse - reverse playback. This item is used to "reverse" your sound fragment, i.e. to play it from the end to the beginning. When this item is checked, the mode is on. Input is the input of the sound. When you select this item, the recording of the sound into the memory begins immediately. So, before you do this, check your boombox and the cords connecting it to your computer.

Double Beep - Double VER. The name of this item speaks for itself - it is an analogue of the BASIC VER operator, but plays two frequencies simultaneously. Frequency 1:XXX and Frequency 2:XXX are the first and second frequencies, respectively. Duration:XXX is the duration of the effect. The M ain M enu, Play and Save items are described above.

Exploding I - explosion simulator 1. This effect is a noise with a smoothly changing frequency. In addition to the three standard functions, you will find the following in this window: Frequency:XXX - initial frequency of the effect. Duration:XXX - duration of one step of the frequency. Length:XXX - effect's length. Group:XX - this parameter influences the sound a little. Its value defines ROM address, from which the data for noise generation is taken. Increase - if you select this item, it will change its name to Decrease. If you think this is not enough, and you select this option again, it will change back to Increase. This option determines the direction in which the noise frequency is shifted.

Exploding 2 - simulated explosion 2. This effect is similar to the previous one, but the moments of frequency change are much more noticeable. When editing this effect, the only thing you might have trouble with is Length:XXX, which is the length of the whole effect. The others (Frequency:XXX, Increase, M ain M enu, Play and Save) are completely similar to the same ones described above.

Volume FX is a volume effect. This is one of the main attractions of SUPER SOUND. This item of the main menu allows you to create effects with different volume and even change it during playback. The Frequency:XXX, Duration:XXX and Length:XXX items are exactly the same as in Exploding 1. The Volume:XX item sets the initial volume of the effect.

The following three points should be explained in more detail. These are: FQ:Increase, DR: Increase and VL:Decrease. These items define the change in frequency, duration, and volume, respectively. They can take the values Increase, Decrease and No change. Note that because of the way this effect is organized, if you set a volume change, it must be compensated by a frequency change in the opposite direction. Otherwise, that frequency will shift. The last three points of this effect are standard (see above).

Flowing 1 - smooth effect 1. This effect is a tone with a smoothly changing frequency. All items in this window are similar to those in Exploding 1, so I will only list them, without descriptions: Frequency:XXX, Duration:XXX, Length: XXX, Increase (Decrease), M ain M enu, Play and Save.

Flowing 2 - smooth effect 2. This effect is more sophisticated than the previous one. The frequency of the tone at a particular moment in time is made up of two components, which can change independently of each other. In the menu, these components are named Frequency I :XXX and Frequency 2:XXX, and the ways of changing them are Fl:<mode> and F2:<mode>, where <mode> can take the values Increase, Decrease and No change (see above). The Duration:XXX item is completely similar to the one in Exploding I, and M ain M enu, Play and Save are standard.

Cycle 1 is Cycle effect 1. This and the following effects are based on the VEER subroutine from the ZX-Spectram ROM. Cycle 1 is very similar in properties to Flowing 1, but its timbre is much different. The items Frequency:XXXX, Duration:XXXX, Increase (Decrease), M ain M enu, Play and Save are similar to those in Flowing 1. The Quantity:XXX item determines the number of passes in the loop. It is similar to the Length:XXX parameter of the previous effects.

The last unknown item is Step:XXX. It defines the step of frequency change.

Cycle 2 - Cycle effect 2. This effect differs from Cycle 1 only in that the frequency change is limited and when crossing a certain threshold its value becomes close to the initial one, which allows you to create quite interesting variations. All items of this effect are completely similar to those in Cycle 1.

Noise I - Noise 1. This effect is ordinary noise. The items Frequency:XXX, Length:XXXX, Group:XX, M ain M enu, Play and Save are the same as those in Exploding 1. Bounds - Bounces. If this item is checked, the noise becomes sort of bouncing.

Noise 2 - Noise 2. This is the noise analogue of Flowing 2, so you can find descriptions of all items of this effect in Flowing 2. The only exception is Group:XX, its description is in Exploding 1.

7.1.2. Using Effects.

Now a little about using uploaded effects. Any of them can be loaded into memory by typing from the keyboard

LOAD "name" CODE addr for the tape and

RANDOMIZE USR 15619: REM: LOAD "name" CODE addr for disk

- where name is the name of the file on the tape or disk, and addr is the address where you want to load the effect. To run the loaded effect, type

RANDOMIZE USR addr.

Note that the files recorded from the Input Sound window are usually very large, so make sure that they do not overlap the system RAM areas when they are loaded.

7.1.3. Versions.

There are several different versions of SUPER SOUND:

1.0 - This is the trial version. It suffers from almost every flaw imaginable and it is better for no one to see it.

2.0 - The program has been significantly improved. It contains a dynamic screen saver. This version served as the basis for all subsequent versions.

2.1 - Removed the screensaver, added range control to Cycle 1, and a new effect, Volume FX.

2.2 - Improved design, fixed minor bugs in the playback subroutines.

The release of version 3.0, which will be an order of magnitude better than its predecessors, is planned. It is supposed to expand the Information, Input Sound and Double Beep features, add a compiler and the ability to create effects for the coprocessor, several new effects and useful functions.

Version 1.0 is only available in cassette. Versions 2.0 and 2.1 are available in both cassette and disc versions. Version 2.2 is disk only. Version 3.0 will be for Spectrum 128K and some functions will not work on computers with less memory.

7.2. W ham the M usic Box music editor.

W ham the M usic Box is probably the richest music editor for ZX-Spectram. With its help you can create two-voice melodies with a drum and noise effects. The created musical fragment can be saved on tape or floppy disk, and if necessary, you can load it again and make changes. A great advantage of the editor is that melodies written with it can be used for the design of other programs.

After the W ham is loaded, a melody is played and the main menu (M AIN M ENU) appears on the screen. To select a mode, press the corresponding numeral key:

1. LOAD TUNE - loading a tune
2. SAVE TUNE to save the tune
3. HEAR TUNE - to listen to a tune
4. W HA M PILER - compilation

5. SET TE M PO - tempo setting
6. EDIT M ODE - editing mode
7. HELP PAGE - hint

7.2.1. LOAD TUNE - to load a tune.

This mode allows you to load a file containing a previously created melody into the editor for further work with it. When you enter LOAD TUNE mode (key 1), the program will ask you from which device you want to load the file:

SELECT PERIFERAL DEVICE
MEMORY DRIVE CONTAINER (SPACE=EXIT)

If you decide to give up on your idea, press the spacebar. Otherwise, select the device with one of the keys: T - cassette, M - RAM, D - microdrive.

When loading from a cassette tape, when prompted "FILENA M E ?" enter the name of the file in which the tune is stored.

With W ha m comes a file called EXA M PLE which contains a cheerful tune on a cassette (floppy disk). Try downloading it and listening to it (see next).

Since the microdrive has not caught on in Russia, and the disk drive has caught on instead, there is a legitimate desire to replace the inscription DRIVE with DISK, with all the ensuing consequences. This was quite successfully done by several Russian programmers (there is also a version of the program in Russian). The most common disk version before the request "FILENA M E ?" (see above) asks if it should print the directory of disk "A" (Yes/No):

PRINT CATALOGUE 'A:' (Y/N) ?

And then it loads in the standard way.

If you access the RAM (key M), a list of six melodies will appear on the screen:

1. FREEDO M
2. TROPIC ANA
3. YOUNG GUNS
4. W HISPER
5. BAD BOYS
6.

The first five are created by the authors of the program, and the sixth is reserved for your creations. To load any of them, just press the corresponding number key.

When the download is complete, the program automatically enters M AIN M ENU mode.

7.2.2. SAVE TUNE - to save the tune.

Thanks to this mode your work will not be wasted when you have to break away from composing a melody for a long time, that is, the created musical fragment can be saved on tape or disk for further improvement.

You can enter SAVE TUNE mode from the main menu by pressing the 2 key, after which the familiar question about the device to which you wish to record your work will appear:

SELECT PERIFERAL DEVICE
MEMORY DRIVE CONTAINER (SPACE=EXIT)

As before, pressing the space bar will take you back to the main menu, and selecting the device: T - cassette, M - memory and D - microdrive (diskette) will prompt "FILENA M E ?" to which you must enter the name of the file in which the tune will be saved.

If you choose to record your piece of music to RAM, you will see a list of six melodies in front of you. Your song can be recorded in place of any of them, but it will be stored only until you turn off the power or press the reset button.

In the case of a tape recording, the question appears after it is finished: VERIFY (Y/N). If you are sure of the quality of the recording you can press "N", otherwise press "Y",

rewind the tape to the beginning of the file you just recorded and play it back to check it.

When the recording is complete, the program automatically enters M AIN M ENU mode.

7.2.3. EDIT MODE is ***the editing mode.***

The editing mode is the main tool with which you can realize your musical ideas.

You can enter this mode from the main menu by pressing the 6 key. This will display images of the two stanzas (for bass and treble keys) as well as the octave indicator OCTAVE, the current channel (voice) indicator CHANNEL 1 (2) and the step counters for each of the channels COUNTERS (CHN1, CHN2).

The top two rows of keys are used to select the editor functions, and the bottom two rows replace the piano keyboard. You can see the meanings of these keys in Fig. 5. The row from Caps Shift to Space acts as white keys, and from A to L as black keys.

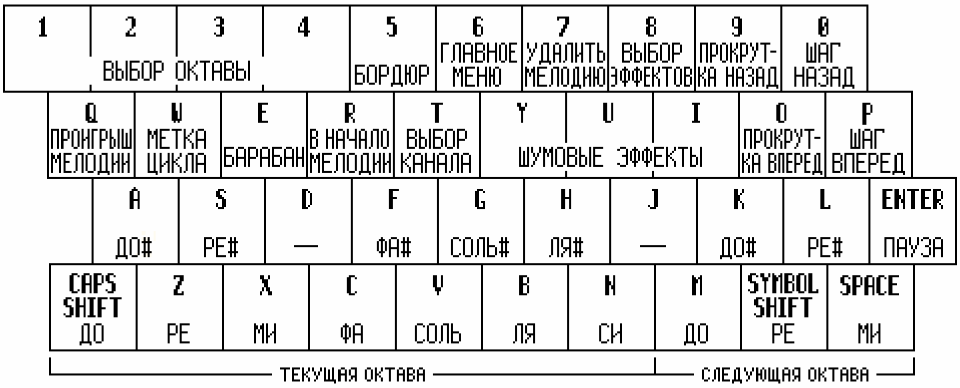


Рис. 5 Функции клавиш в режиме редактирования.

So, you want to "sketch" a melody. Pressing the key T, select the voice you are going to work with with keys 1 ... 4 - octave (1 - large, 2 - small, 3 - first, 4 - second) and try to "play" a melody on the "sound" keys. At the same time on the stave will appear images of notes, and colored squares indicate the corresponding keys of the piano (for the first voice the square is red, for the second - purple). If you pressed the wrong key by mistake, don't get upset, everything can be corrected: pressing 0 will take you one step back. If you need to go back more than one step, press 9.

Press the P key to play one note one step forward. By holding down the P key you can listen to a part of the melody. With the key O you can "fast forward" the melody.

You might want to listen to the entire piece of music without exiting edit mode. Press R (return to the beginning of the tune), then Q (play the tune), and your creation will play. You can stop playback by pressing any key.

To write the second voice, switch the T key to the channel and use the R, O, P, 0, 9 keys to go to the step from which you are going to use it.

Unfortunately, the W ha m does not allow you to record notes of different durations. You can achieve the desired duration either by using the Enter key (pause) or by recording the desired number of equal notes.

Your music piece will sound very different if you use a drum (the E key) and noise effects (the Y, U, I keys) to arrange it. The drum can sound differently, depending on which channel it is set in. In the first channel, it is really a drum, and in the second channel, it sounds more like a note of the C octave or DO of the large octave. Noise effects can be edited using the appropriate mode, which can be accessed from EDIT M ODE by pressing the 8 key. After that, a menu for selecting the frequency (W AVEFOR M) and duration (DURATION) of the noise effects will appear on the screen. To select one

from the 16 options for duration and 8 options for frequency (form of oscillation) of the noise effects provided by the program, you can use the following keys:

5,8 - Selects the noise effect to edit;
6 - Switching to the frequency selection function;
7 - Switching to the duration selection function;
0 - Changing the effect.

You can listen to the resulting editing noise effects using the 9 key.

After adjusting the effects, you can exit back to the melody editing mode (any key except those involved in the effects setup) and use them in your work with the Y, U and I keys. Although only these three keys are dedicated to noise effects, you can use a much larger number of them (up to 128). To do this, you create the first three effects and place them in the desired locations in the tune, then the next three, and so on.

The presence of the drum and noise effects in the tune is displayed on the stanza as a space, which is very inconvenient. Note that a note cannot be sounded at the same time as a drum and noise effects (the only exception is a drum set in the second channel).

If you want the melody to repeat from any step, i.e. to sound continuously, use the W key to set the cycle mark on the selected step in both channels. The computer will ask:

SET LOOP HERE ? (Y/N) - set mark here ? (Yes/No)

The positions of the marks in the first and second channels may not coincide. But in order not to disturb the synchronism of voices during the cyclic playback of the melody, the number of steps in the first and second channels should be a multiple of each other. For successful compilation (see below) it is obligatory to set the label in both channels.

If you want to clear the memory of an unsuccessful work, press 7 and confirm the request:

ERASE CURRENT TUNE ? (Y/N) - erase current tune ? (Yes/No)

With key 5 you can change the color of the border. And to return to the main menu, press key 6.

7.2.4. HELP PAGE - hint.

In the editing mode you need to use a lot of keys, it is not easy to remember their functional purpose at once. Therefore as a help you are offered a hint page - HELP PAGE, which you can call from the main menu by pressing the key 7.

A list of keys used for editing appears on the screen:

KEYS	FUNCTION	
1-4	SELECT OCTAVE	octave selection
5	CHANGE BORDER	border color change
6	M AIN M ENU	main menu access
7	ERASE TUNE	ringtone deletion
8	NOISE M IXING	editing noise effects
9	REPEAT BACKSTEP	scrolling back
O	SINGLE BACKSTEP	one step back
Q	REPLAY TUNE	tune playback
E	DRU M EFFECT	reel
T	CHANGE CHANNEL	channel selection
Y,U & I	NOISE EFFECTS	noise effects
O	FAST FOR W ARD	scrolling forward
P	PLAY NEXT NOTE	play the next note

The lower half of the keyboard mimics a piano, enter is a rest

The lower half of the keyboard mimics a piano, enter is a rest

Use cursor keys to edit the Use cursor keys to edit the white noise with 9 to hear the white noise, current effect and 0 to alter it 9 to hear, 0 to change

7.2.5. HEAR TUNE - **to listen to the tune.**

When you listen to a melody in EDIT M ODE mode, it sounds somewhat distorted, because the processor is busy playing music, displaying notes, handling the keyboard, etc. To get an idea of the true sound of the tune, that is, to hear how it will be played after compilation (see below), use the HEAR TUNE function (key 3 of the main menu).

After listening, pressing any key will take you to EDIT M ODE mode.

7.2.6. SET TEMPO - **set the tempo.**

If you want to change the tempo after listening to a tune, use the SET TE M PO mode (main menu key 5).

You can use key 8 to speed up the melody, and key 5 to slow it down. The change in tempo is shown visually with a purple bar on the screen. As the tempo decreases, the bar also decreases and vice versa.

Press any key except 5 and 8 to enter edit mode (EDIT M ODE).

7.2.7. WHAMPILER - **compilation.**

A piece of music saved as an editable file in the SAVE TUNE mode can be loaded again (LOAD TUNE) and continued. However, this way of presenting the tune allows you to listen to it only when you are in the editor.

If you want to make music in a program written in BASIC or Assembler, you must write the melody as a program that works independently of the editor. Such a subroutine is created using the W HA M PILER compilation mode.

The resulting compilation writes a file to tape or floppy disk which is a program in codes that runs separately from the W ham, but, unlike the edited file, cannot be further modified.

Before compiling, use the LOAD TUNE function to load the tune you want to compile and check whether the loop marks are set in both channels.

As an example, let's try to compile one of the 5 melodies loaded from memory (see LOAD TU NE - M E M ORY), let's say theme number 1 - FREEDO M . Load it and, by pressing key 4, enter compilation mode. When prompted by TUNENA M E ? enter the name you want to save the compiled file under, for example FREE- M US. Next, the program will ask you to enter the address from which the compiled file will be located and run:

ASSEMBLY ADDRESS ?

Specify a decimal address of at least 32768, such as 60000, and then the compiler will display the following information:

TUNE NAME:	FREE-MUS name of the tune
ADDRESS: 60000	address:
RETURN OPTION:	KEYPRESS return condition
WHITE NOISE: --NONE--	no noise effects CHANNEL 1
LENGTH: 313	number of steps in channel 1
CHANNEL 1 LOOPSTART	loop from beginning in channel 1
CHANNEL 2 LENGTH: 313	number of steps in channel 2 CHANNEL 2
LOOPSTART	loopfrom beginning in channel
2	

1. KEYPRESS
2. ALWAYS
3. TUNEEND

RETURN OPTION 1, 2 OR 3 ?

The numbers 1, 2, and H represent the end of tune playback. Depending on the option you select, certain modifications of the compiled file are obtained:

KEYPRESS - Playing a tune ends when any key is pressed;

AL W AYS - the melody is played simultaneously with the operation of the program (this mode is described in detail below);

TUNEEND - playback ends either when the tune ends or when any key is pressed.

The compilation will start as soon as you select one of the listed conditions with keys 1, 2 or 3. Press the 1 key, for example.

When the compilation is complete, the executable file length in bytes will be displayed and a message about the normal completion of the operation will be displayed:

CODE LENGTH: 893

COMPILATION COMPLETED OK

Then, after the inscription ADJUST M ENT POKES: ("setup POKE"), the information you need to configure the executable already when using it in your program will appear (remember that in our example ASSE M BLY ADDRESS is 60000):

REPLAY SPEED: 60035, (230 TO 255)

- This tells you that by writing a number between 230 and 255 to memory location 60035 (ASSE M BLY ADDRESS + 35), you can change the tempo of the tune;

BORDER COLOR: 60026, (0 TO 7)

- this message tells you that by writing a number from 0 to 7 to memory cell 60026 (ASSE M BLY ADDRESS + 26), you can set the desired border color (purple by default) for the duration of the tune;

THEN RUN - RANDOMIZE USR 60000

- this inscription reminds you to use the command RANDO M IZE USR 60000 (ASSE M BLY ADDRESS) to start the tune from the BASIC.

After these messages are displayed, the executable file is written to a cassette or disk, depending on the version.

Now you can safely press the reset button and load the compiled program into memory in codes. If the executable file is saved on the tape, write the following BASIC program to load and run it:

```
10 LOAD "FREE-MUS" CODE
20 POKE 60026.7: REM border color - white
30 RANDOMIZE USR 60000
```

Start it up and load the tune, and then you'll hear the compiled tune playing until you press a key. Experiment with the tempo of the piece by using the ROKE operator to enter various numbers between 230 and 255 in cell 60035.

For the disk version, replace the first line of the program with the following:

```
10 RANDOMIZE USR 15619: REM: LOAD "FREE-MUS" CODE
```

Now you can compile any melody (including your own) by following the same steps.

The compilation process may go a little differently. For example, if you decided to make a repeating bass part and looped it much earlier than the main melody (make sure that the number of steps in both channels is a multiple of each other). In this case the program eliminates the possibility of compilation in the TUNEPEND mode and sets the KEYPRESS mode, and instead of the possibility to select asks:

CHANGE RETURN OPTION TO ALWAYS ?

change the compilation mode to ALWAYS ?

If you press the Y key, the mode will be changed. Otherwise the compilation will continue in KEYPRESS mode.

Note that the compiler outputs the number of steps in the channel along with the loop label. So you may find that, for example, 65 is a multiple of 9 (in fact, it is 64 and 8).

If in your melody the number of steps in both voices is not a multiple of each other, the compilation will be interrupted by the next message:

```
WARNING                            WARNING
End markers mismatchHe markers match
cycle. may cause distorted tune May distorttune. Continue
?(Y/N)                             Continue ? (Yes/No)
```

If you press Y, the compilation will continue as if the number of steps were a multiple of each other, but no one is responsible for what happens. Otherwise the compilation will stop and you will end up in the main menu.

If you forget to place the cycle marker in at least one of the channels, you will see the following picture:

```
       TUNE NAME: XXXXXXXXXX
ASSEMBLY ADDRESS: XXXXXX
   RETURN OPTION: KEYPRESS
     WHITE NOISE: XXXXXXXX
CHANNEL 1 LENGTH: NOT PRESENT
the number of steps in the first channel is undefined
CHANNEL 1 LOOP : START
CHANNEL 2 LENGTH: NOT PRESENT
the number of steps in the second channel is undefined
CHANNEL 2 LOOP : START

Tune                               compilation abandoned.
No end markers defined.            No end markers defined Use the
W key to place the end             Usethe W keyplace a marker for the
current channel.                   marker in each channel
```

If the machine scolded you, take it easy. Press Enter, exit the compilation mode, fix the defects you noticed and be sure that on the second or at least the third try the melody will be compiled and you will be able to decorate your program with it. Most likely, you will want your program to run at the same time as playing the melody. You can achieve a similar effect if you use AL W AYS mode when compiling. A file compiled in this mode is called at two addresses: it has two entry points. The first is used to initialize the melody and play the first note, and is located at the same address as in other modes (ASSE M BLY ADDRESS), and the second is used to play all subsequent notes (located at ASSE M BLY ADDRESS + 12).

So if you use the AL W AYS mode in the above compilation example, then to initialize the melody and play the first note you need to execute the operator

RANDOMIZE USR 60000,

and to play each successive note, the operator

RANDOMIZE USR 60012.

Here is an example of a program that demonstrates the capabilities of AL W AYS mode:

```
10 PAPER 0: BORDER 0: INK 7: BRIGHT 1: CLS
20 LOAD "FREE-MUS" CODE: CLS: OVER 1
30 LET A$="ALWAYS-mode demonstration!"
40 RANDOMIZE USR 60000
50 PRINT AT 10.3;
60 FOR A= 1 TO LEN A$: RANDOMIZE USR 60012
70 PRINT INK RND*6+1; A$(A);
80 NEXT A: GOTO 50
```

You can do the same for programs in machine code. Note that the execution time of your program between notes should not be very long, otherwise the quality of the melody will be lost. In addition, it should be about the same between any two notes.

7.2.8. Tips.

1. If the program has gone to the BASIC with an error message (for example, when loading from a tape recorder or if you pressed the BREAK key completely by accident), enter the RUN command to return to the main menu.
2. If you decide to create a less lengthy melody using both voices, you must introduce these voices in parallel, otherwise errors are inevitable. Also, it's a good idea to listen to what you've got from time to time.
3. You should save all the melodies you compose on a single cassette (or diskette) using the SAVE TUNE function. This ensures that after a while you will have a fairly large selection of tunes to choose from when designing a new program.

Appendix 1 .

Listings of SUPER SO UNDa sound effects.

All effects are listed exactly as in SUPER SOUND v2.2 and are provided with comments. Labels starting with "+", "*", ":" and "?" are special and mark the values to be changed:

+ – one-byte changeable value
* – two single-byte modifiable values
: – two-byte variable value
? – variable command

The word following the special symbol repeats the name of the item in the program that is responsible for this parameter (possibly in abbreviated form). In the case of a changeable command, the comment (in brackets) specifies all possible variants. For normal compilation, special marks must be removed or renamed.

Input Sound:

```
10              DI                       ; interruption ban
20              LD     HL,37000          ; HL=save audio address
30 :LENGTH      LD     DE,27000          ; DE=duration of sound
40 LOOP1        LD     B,8               ; B= bit counter
50 LOOP2        SLA    (HL)              ; scrolling the value in memory
60              IN     A,(254)           ; enter a value from port 254
70              BIT    6,A               ; check the tape recorder bit
80              JR      Z,+INPUT_SPD     ; if 0, then switch to NOSIGN
90              SET    0,(HL)            ; set bit D0 in memory
100 +INPUT_SPD  LD     C,3               ; C=delay
110 PAUSE       DEC    C                 ; C=C-1
120             JR     NZ,PAUSE          ; If C<>0, the loop
130             DJNZ   LOOP2             ; continue the byte cycle
140             INC    HL                ; HL=HL+1
150             DEC    DE                ; DE=DE-1
160             LD     A,D               ; DE=
170             OR     E                 ; 0 ?
180             JR     NZ,LOOP1          ; If not, the cycle
190             EI                       ; interrupt resolution
200             RET                      ; Returns
```

Play Sound

```
10              CALL   124               ; RET is located at address 124
20              DEC    SP                ; raise the stack pointer
30              DEC    SP                ; by two bytes
40              POP    HL                ; remove from the stack the address of
                                           line 20
50              LD     DE,47             ; DE=number of bytes from line 20 to
                                           the buffer
60              ADD    HL,DE             ; HL=buffer address
70 :LENGTH      LD     DE,27000          ; DE=duration of sound
80 ?REVERSE     NOP                      ; reserve for REVERSE (ADD HL,DE)
90              DI                       ; interruption ban
100             LD     A,(23624)         ; A=
110             RRA                      ; color
120             RRA                      ;  bor-
130             RRA                      ;    dura
140 LOOP3       LD     B,8               ; B= bit counter
150 LOOP4       AND    239               ; reset of bit D4 of register A
160             RLC    (HL)              ; scrolling data through the CY flag
170             JR     NC,NOSGN          ; If the CY=0 flag, switch to NOSGN
180             OR     16                ; setting of bit D4 of register A
190 NOSGN       OUT    (254),A           ; output A to port 254
200 +PLAY_SPD   LD     C,3               ; C=delay
210 PAUS2       DEC    C                 ; C=C-1
```

```
220              JR    NZ,PAUS2       ; If C<>0, the loop
230              DJNZ  LOOP4          ; continue the byte cycle
240  ?REVERSE    INC   HL             ; HL=HL+1 (DEC HL)
250              LD    C,A            ; preservation A
260              DEC   DE             ; DE=DE-1
270              LD    A,D            ; DE=
280              OR    E              ; 0 ?
290              LD    A,C            ; Recovery A
300              JR    NZ,LOOP3       ; If DE<>0, the loop
310              EI                   ; interrupt resolution
320              RET                  ; Returns
```

Double Beep

```
10               DI                   ; interruption ban
20  *FRQ1,FRQ2   LD    DE,13000       ; E=frequency 1, D=frequency 2
30  +DURATION    LD    H,100          ; H=duration
40               LD    A,(23624)      ; A=
50               RRA                  ; color
60               RRA                  ;  bor-
70               RRA                  ;     dura
80               LD    C,A            ; preservation A
90               EX    AF,AF'         ; change the A and F registers to
                                        alternate registers
100              LD    A,C            ; Recovery A
110              LD    C,E            ; C=counter 1
120              LD    B,D            ; B=counter 2
130  BEEP        EX    AF,AF'         ; changing A and F registers (voice
                                        change)
140              DEC   C              ; C=C-1
150              JR    NZ,CONT        ; If C<>0, change to CONT
160              LD    C,E            ; Restore counter 1
170              XOR   16             ; Invert bit D4 voice 1
180  CONT        OUT   (254),A        ; output A to port 254
190              EX    AF,AF'         ; changing A and F registers (voice
                                        change)
200              DEC   B              ; B=B-1
210              JR    NZ,CONT2       ; if B<>0, then go to CONT2
220              LD    B,D            ; Restore counter 2
230              XOR   16             ; Invert bit D4 of voice 2
240  CONT2       OUT   (254),A        ; output A to port 254
250              INC   L              ; L=L+1
260              JR    NZ,BEEP        ; if L<>0, switch to BEEP
270              DEC   H              ; H=H-1
280              JR    NZ,BEEP        ; if H<>0, switch to BEEP
290              EI                   ; interrupt resolution
300              RET                  ; Returns
```

Exploding 1

```
10  :GROUP       LD    HL,0           ; HL = ROM address
20               LD    A,(23624)      ; A=
30               RRA                  ; color
40               RRA                  ;   bor-
50               RRA                  ;     dura
60               LD    D,A            ; D=A
70  *FRQ,LEN     LD    BC,38401       ; C=frequency,B=length
80  LOOP1        PUSH  BC             ; saving BC
90  +DURATION    LD    B,20           ; B=duration
100  LOOP2       LD    A,(HL)         ; A=the contents of the
                                        ROM cell
110              AND   248            ; reset the curb bits
120              OR    D              ; installing curb bits
130              OUT   (254),A        ; output A to port 254
140              PUSH  BC             ; saving BC
150              LD    B,C            ; B=frequency
```

```
160 LOOP3       DJNZ  LOOP3         ; delay
170             INC   HL            ; HL=HL+1
180             POP   BC            ; BC recovery
```

```
190              DJNZ  LOOP2         ; cycle
200              POP   BC            ; BC recovery
210 ?INCREASE    INC   C             ; increase in delay (DEC   C)
220              DJNZ  LOOP1         ; cycle
230              RET                 ; Returns
```

Exploding 2

```
10               LD    A,(23624)     ; A=
20               RRA                 ; color
30               RRA                 ;  bor-
40               RRA                 ;    dura
50               LD    L,A           ; L=A
60 *FRQ,LEN      LD    DE,12801      ; E=frequency,D=length
70 LOOP1         PUSH  DE            ; save DE
80 LOOP2         LD    B,E           ; B=E
90 PAUSE         DJNZ  PAUSE         ; delay
100              LD    A,(BC)        ; A=the contents of the
                                       ROM cell
110              AND   248           ; reset the curb bits
120              OR    L             ; set the color of the
                                       border
130              OUT   (254),A       ; output A to port 254
140              INC   C             ; ROM address increment
150              INC   D             ; D=D+1
160              JR    NZ,LOOP2      ; If D<>0, the loop
170              POP   DE            ; rebuild DE
180 ?INCREASE    INC   E             ; increase in delay (DEC   E)
190              DEC   D             ; D=D-1
200              JR    NZ,LOOP1      ; If D<>0, the loop
210              RET                 ; Returns
```

Volume FX

```
10               DI                  ; interruption ban
20 *FRQ,VOL      LD    BC,12900      ; C=frequency,B=volume
30 *LEN,DUR      LD    DE,25650      ; E=length,D=duration
40               LD    A,(23624)     ; A=
50               RRA                 ; color
60               RRA                 ;  bor-
70               RRA                 ;     dura
80 LOOP1         PUSH  DE            ; DE retention
90 LOOP2         PUSH  BC            ; saving BC
100              XOR   16            ; inverting the D4 bit
110              OUT   (254),A       ; output A to port 254
120 LOOP3        DJNZ  LOOP3         ; delay (volume)
130              XOR   16            ; inverting the D4 bit
140              OUT   (254),A       ; output A to port 254
150              LD    B,C           ; B=C
160 LOOP4        DJNZ  LOOP4         ; delay (frequency)
170              POP   BC            ; BC recovery
180              DEC   D             ; D=D-1
190              JR    NZ,LOOP2      ; If D<>0, the loop
200              POP   DE            ; DE recovery
210 ?DR:INC.     INC   D             ; increase in duration (DEC    D,NOP)
220 ?VL:DEC      DEC   B             ; volume reduction (INC B,NOP)
230 ?FQ:INC.     INC   C             ; frequency increase (DEC C,NOP)
240              DEC   E             ; E=E-1
250              JR    NZ,LOOP1      ; if E<>0, the loop
260              EI                  ; interrupt resolution
270              RET                 ; Returns
```

Flowing 1

```
10               DI                  ; interruption ban
```

```
20              LD    A,(23624)     ; A=
30              RRA                 ; color
```

```
40              RRA                    ;  bor-
50              RRA                    ;     dura
60 *FRQ,LEN     LD    BC,65281         ; C=frequency,B=length
70 LOOP1         PUSH  BC              ; saving BC
80 +DURATION  LD      B,20             ; B=duration
90 LOOP2        XOR   16               ; inverting the D4 bit
100             OUT   (254),A          ; output A to port 254
110              PUSH  BC              ; saving BC
120             LD    B,C              ; B=C
130  LOOP3       DJNZ  LOOP3           ; delay
140             POP   BC               ; BC recovery
150              DJNZ  LOOP2           ; cycle
160             POP   BC               ; BC recovery
170  ?DECREASE  DEC   C                ; frequency reduction     C)
                                         (INC.
180              DJNZ  LOOP1           ; cycle
190             EI                     ; interrupt resolution
200             RET                    ; Returns
```

Flowing 2

```
10              DI                     ; interruption ban
20 *FRQ1,FRQ2   LD    DE,2660          ; E=frequency 1,D=frequency 2
30 +DURATION    LD    C,255            ; C=duration
40              LD    A,(23624)        ; A=
50              RRA                    ; color
60              RRA                    ;  bor-
70              RRA                    ;    dura
80 LOOP1        XOR   16               ; inverting the D4 bit
90              OUT   (254),A          ; output A to port 254
100             LD    B,D              ; B=D
110  LOOP2      DJNZ  LOOP2            ; delay 1
120             XOR   16               ; inverting the D4 bit
130             OUT   (254),A          ; output A to port 254
140             LD    B,E              ; B=E
150  LOOP3      DJNZ  LOOP3            ; delay
160  ?F1:INC.   INC   D                ; increase in delay 1 (DEC D,    NOP)
170  ?F2:DEC    DEC   E                ; increase in delay 2 (INC E,    NOP)
180             DEC   C                ; C=C-1
190             JR    NZ,LOOP1         ; If C<>0, the loop
200             EI                     ; interrupt resolution
210             RET                    ; Returns
```

Cycle 1

```
10 +QUANTITY    LD    B,24             ; B=number of notes
20 :FREQUENCY   LD    HL,200           ; HL= initial frequency
30 :DURATION    LD    DE,1             ; DE=duration
40              PUSH  HL               ; HL preservation
50              PUSH  BC               ; saving BC
60              CALL  949              ; ROM subroutine call
70              POP   BC               ; BC recovery
80              POP   HL               ; HL reconstruction
90 :STEP        LD    DE,30            ; DE=frequency step
100 ?           ADC   HL,DE            ; HL increase (SBC HL,DE)
110             DJNZ  :DURATION        ; cycle
120             RET                    ; Returns
```

Cycle 2

```
10 *STP,QUANT   LD    BC,12870         ; C=step,B=number of
                                         notes
20 :FREQUENCY   LD    HL,512           ; HL= initial frequency
30 :DURATION    LD    DE,20            ; DE=duration
40              PUSH  HL               ; HL preservation
```

```
50              PUSH  BC            ; saving BC
60              CALL  949           ; ROM subroutine call
```

```
70              POP   BC            ; BC recovery
80              POP   HL            ; HL reconstruction
90              LD    A,L           ; reduce-
100 ?DECREASE   SUB   C             ; (ADD A,C)
110             LD    L,A           ; L
120             DJNZ  :DURATION     ; cycle
130             RET                 ; Returns
```

Noise 1

```
10 :GROUP       LD    HL,0          ; HL = ROM address
20              LD    A,(23624)     ; A=
30              RRA                 ; color
40              RRA                 ;  bor-
50              RRA                 ;    dura
60              LD    D,A           ; D=A
70 :LENGTH      LD    BC,2560       ; BC=duration
80 BEGIN        PUSH  BC            ; saving BC
90              LD    A,(HL)        ; A=the contents of the
                                      ROM cell
100             AND   248           ; reset the curb bits
110             OR    D             ; border color setting
120             OUT   (254),A       ; output A to port 254
130 +FREQUENCY  LD    B,40          ; B=frequency
140 LOOP        DJNZ  LOOP          ; delay
150 ?BOUNDS     INC   HL            ; HL=HL+1 (INC L)
160             POP   BC            ; BC recovery
170             DEC   BC            ; BC=BC-1
180             LD    A,B           ; BC=
190             OR    C             ; 0 ?
200             JR    NZ,BEGIN      ; If not, the cycle
210             RET                 ; Returns
```

Noise 2

```
10 :GROUP       LD    HL,0          ; HL = ROM address
20 *FRQ1,FRQ2   LD    DE,2660       ; E=delay 1,D=delay       2
30 +DURATION    LD    C,255         ; C=duration
40              LD    A,(23624)     ; A=
50              RRA                 ; color
60              RRA                 ;  bor-
70              RRA                 ;    dura
80 BEGIN        XOR   16            ; inverting the D4 bit
90              OUT   (254),A       ; output A to port 254
100             LD    B,(HL)        ; B=content of the ROM
                                      cell
110 LOOP1       DJNZ  LOOP1         ; delay 1
120             LD    B,D           ; B=D
130 LOOP2       DJNZ  LOOP2         ; delay 2
140             XOR   16            ; inverting the D4 bit
150             OUT   (254),A       ; output A to port 254
160             LD    B,(HL)        ; B=content of the ROM
                                      cell
170 LOOP3       DJNZ  LOOP3         ; delay 3
180             LD    B,E           ; B=E
190 LOOP4       DJNZ  LOOP4         ; delay 4
200             INC   HL            ; HL=HL+1
210 ?F1:INC     INC   D             ; increase D (DEC D,NOP)
220 ?F2:DEC     DEC   E             ; reducing E (INC E,NOP)
230             DEC   C             ; C=C-1
240             JR    NZ,BEGIN      ; If C<>0, the loop
250             RET                 ; Returns
```

Appendix 2.

Tips for using the assembler.

To test the effects in machine code given in this book, you will have to use a special program - an assembler. Nowadays there are many different assemblers, but in Russia GENS4 is the most popular and widespread. It is a part of package DEVPAC4 by Hisoft. The second part of this package (the monitor-debugger M ONS4) does not interest us now.

I will not describe GENS4 in detail, it is well done in [1], but I will explain some key points.

There are several different versions of this program (including disk versions), so I can not advise anything specific about the download, except that for safety you can skip the base-loader and load the code part directly. To load it from the tape, enter the following commands:

CLEAR 24999: LOAD "" CODE 25000

And from the disk, it's like this:

CLEAR 24999: RANDOMIZE USR 15619: REM: LOAD "GENS4D" CODE 25000

To start the GENS4 loaded in this way, enter the command

RANDOMIZE USR 25000.

Almost all of the effects in this book can be entered and assembled without any changes. The exceptions are SUPER SOUND effect listings, where you need to replace special labels with more acceptable ones (they should start with a Latin letter, be somewhat shorter and not contain spaces).

Before any effect it is necessary to insert an ORG assembly directive specifying from which address to place the translated code. In this case this address must be at least 37000. For example:

```
10ORG 40000
```

If you want to call the created effect directly from the assembler, you have to insert the ENT directive:

```
20ENT $
```

Now a little about the commands of the GENS 4 line editor (you can find their complete list in [1]):

I[N][, M]_ - Automatic numbering of lines starting from line N and in increments of M . Canceling - the Edit key (CS/1). By default, N and M are 10.

L[N] [,M] - Display the program listing from line N to line M inclusive. By default the listing of the whole program is displayed.

D N, M - Delete rows from N to M inclusive.

G[,,S]. - Load a program named S from the magnetic medium.

P[N][, M][,S] - Write the program from line N to line M inclusive to the magnetic medium under the name S. By default the parameters set by the previous command G or P are used.

O[,,S] - Write the compiled program under the name S to the magnetic medium.

R - Run the compiled program. Return to GENS4 by RET command.

B - Return to BASIC. You can restart GENS4 in our case with the command RANDO M IZE USR 25000.

Z - Delete program text.

H - Display a list of editor commands.

A - Assemble the program. The format of this command is rather complicated.

It is described in detail in [1]. Here you just need to enter the letter A without parameters.

List of references.

1. Larchenko A.A., Rodionov N.Y. ZX-Spectram and TR-DOS for users SPb: Peter, 1994.
2. Basics dialects for ZX-Spectram. Edited by A.A. Larchenko, N.Y. Rodionov.
3. USSR Academy of Sciences, Institute for Informatics Problems. Lushchikhina I.Yu. et al. Organization of Digital Music Interface M IDI. Moscow: Preprint, 1988.
4. M IDI M usical Instrument Digital Interface Specification 1.0 North Hollywood: International M IDI Association (I M A), 1983.
5. Schulze, Hans-Jochen/Engel, Georg. M oderne M usikelektronik, Praxisorientiere Elektroakustik und Gerate zur elektronischen Klangerzeugung. Berlin: M ilitarverlag der DDR (VEB), 1989.